Aula 03 (Prof. Evandro Dalla Vecchia)

IFCE (Prof. Ciência da Computação-Metodologia e Técnicas da Comput) Conhecimentos - 2021(Pós-Edital)

Autor:

Diego Carvalho, Equipe Informática e TI, Evandro Dalla Vecchia Pereira , Pedro Henrique Chagas Freitas, Thiago Rodrigues

Cavalcanti, Raphael Henrique Lacerda

Aula 03 (Prof. Evandro Dalla Vecchia)

20 de Setembro de 2021

Sumário

Apresentação do Curso 2
Apresentação Pessoal 3
PARE TUDO! E preste atenção!! 4
Cronograma de Aulas 5
Introdução ao Estudo de Sistemas Operacionais e Arquitetura de Computadores 7
Considerações Iniciais 7
Arquitetura e Organização de Computadores 7
Processamento de Dados, Organização e Arquitetura 7
Arquiteturas Clássicas 8
Questões Comentadas 10
Arquiteturas de Processadores (RISC e CISC) 15
Conceitos 15
Questões Comentadas 18
Hardware x Software 22
Máquina Multinível 22
Algoritmos e Linguagens de Programação de Alto Nível 24
Teste de Mesa 31
Questões Comentadas 32
Linguagens de Máquina e de Montagem 35
Conceitos 35
Instrução de Máquina 35
Representação da Instrução 36
Modos de Endereçamento 37

Linguagem de Montagem 39
Questões Comentadas 42
Compilador, Montador, Interpretador e Ligador 45
Conceitos 45
Compilador 46
Montador e Ligador 47
Interpretador 48
Questões Comentadas (Bancas variadas) 48
Questões Comentadas (Banca CESPE) 55
Lista de Questões 57
Gabarito 73

APRESENTAÇÃO DO CURSO

Iniciamos nosso **Curso Regular de Sistemas Operacionais e Arquitetura de Computadores** em teoria e questões, voltado para provas **objetivas e discursivas** de concurso público. Tais assuntos são cobrados em diversos concursos em que há vagas específicas para a área de TI.

As aulas em PDF possuem por característica essencial a **didática**. Ao contrário do que encontramos em alguns livros, o curso todo se desenvolverá com uma leitura de fácil compreensão e assimilação.

Além disso, teremos videoaulas! Essas aulas destinam-se a complementar a preparação. Quando estiver cansado do estudo ativo (leitura e resolução de questões) ou até mesmo para a revisão, abordaremos alguns pontos da matéria por intermédio dos vídeos. Com outra didática, você disporá de um conteúdo complementar para a sua preparação. Ao contrário do PDF, evidentemente, **AS VIDEOAULAS NÃO ATENDEM A TODOS OS PONTOS QUE VAMOS ANALISAR NOS PDFS, NOSSOS MANUAIS ELETRÔNICOS**. **Por vezes, haverá aulas com vários vídeos; outras que terão videoaulas apenas em parte do conteúdo. Nosso foco é sempre o estudo ativo!**

APRESENTAÇÃO PESSOAL

Meu nome é Evandro Dalla Vecchia Pereira, sou autor do livro "Perícia Digital - Da investigação à análise forense", Mestre em Ciência da Computação (UFRGS), Bacharel em Ciência da Computação (PUCRS), Técnico em Redes de Computadores (Etcom/UFRGS) e em Processamento de Dados (Urcamp). Perito Criminal na área de Perícia Digital desde 2004 no Instituto-Geral de Perícias/RS. Professor de pós-graduação em diversas instituições, nas áreas de Perícia Digital, Perícia Criminal e Auditoria de Sistemas. Lecionei em cursos de graduação de 2006 a 2017, nas instituições PUCRS, Unisinos, entre outras e sou professor em cursos de formação e aperfeiçoamento de Peritos Criminais, Delegados, Inspetores, Escrivães e Policiais Militares.

No Estratégia Concursos leciono desde o começo de 2018, inicialmente na área de Computação Forense e, na sequência, também assumi as áreas de Arquitetura de Computadores e Sistemas Operacionais, tanto na elaboração de materiais escritos como na gravação das videoaulas.

Deixarei abaixo meus contatos para quaisquer dúvidas ou sugestões. Terei o prazer em orientá-los da melhor forma possível nessa caminhada que estamos iniciando.

Instagram: @profevandrodallavecchia

Facebook: https://www.facebook.com/profevandrodallavecchia

PARE TUDO! E PRESTE ATENÇÃO!!

Hoje eu faço parte de uma equipe SENSACIONAL de professores! Depois de muita luta conseguimos reunir um time de profissionais extremamente QUALIFICADO e sobretudo COMPROMISSADO em fazer o melhor pelos alunos. Para tal criamos um conjunto de ações para nos aproximarmos dos alunos, entendermos suas necessidades e evoluirmos nosso material para um patamar ainda mais diferenciado. São 3 as novidades que gostaria de convidá-lo a conhecer:

Nosso podcast alternativo ... livre, descontraído e com dicas rápidas que todo CANETA PRETA raiz deve ouvir. Já temos alguns episódios disponíveis e vários outros serão gravados nas próximas semanas ... acompanhe em:

http://anchor.fm/estrategia-tech

Nosso grupo do Telegram é um local onde ouvimos os alunos e trocamos ideias com eles. Está crescendo a cada dia. A regra do grupo é: só vale falar sobre concursos. Lá divulgamos nossas aulas ao vivo e falamos sobre os concursos abertos, expectativas de novos concursos, revisões de véspera, e por aí vai...

http://t.me/estrategia_ti

Criamos um perfil no Instagram ... e qual o objetivo? Fazer com que os alunos percam tempo nas redes sociais? Claro que não!! Estamos consolidando diversos posts dos professores! São dicas especiais, um patrimônio que deve ser explorado por todos os concurseiros de TI!

http://instagram.com/estrategiaconcursosti

CRONOGRAMA DE AULAS

Vejamos a distribuição das aulas:

AULAS	TÓPICOS ABORDADOS
Aula 00	Arquitetura e Organização de Computadores (AOC) (Parte 1) - Conjunto de Instruções. Ciclo de Instruções. Arquiteturas clássicas. RISC e CISC. Hardware x Software: Algoritmos e Linguagens de Programação. Máquina multinível. Linguagens de máquina e de montagem. Tradução, compilação e interpretação.
Aula 01	AOC (Parte 2) - Processamento: CPU. Registradores. Endereçamento de Instruções. Barramento. Clock. Pipeline. Multiprocessamento. Memória: Hierarquia de memória. Memória cache. UMA e NUMA. Endereçamento em memória. Armazenamento em memória secundária.
Aula 02	AOC (Parte 3) - Sistemas de numeração. Conversão de bases. Aritmética binária. Representação dos dados. Operações Lógicas e Tabela Verdade. Álgebra booleana. Circuitos lógicos.
Aula 03	Hardware: Componentes de um computador. Principais processadores do mercado. Instalação e manutenção de computadores.
Aula 04	Sistemas Operacionais (SO) (Parte 1): Conceitos de Sistemas Operacionais. Gerência de Processos. Gerência de Entrada/Saída (E/S).
Aula 05	SO (Parte 2): Gerência de Memória. Gerência de Armazenamento (Sistemas de Arquivos).
Aula 06	Sistemas de Arquivos FAT12, FAT16, FAT32, NTFS, EXT2, EXT3, EXT4, BTRFS, XFS, entre outros.
Aula 07	Windows Desktop: instalação e configuração. Configuração e serviços de rede. Administração de usuários, grupos, permissões, controles de acesso. Explorador de arquivos. Gerenciamento de Disco. Prompt de Comando: comandos básicos. PowerShell.
Aula 08	Windows Server: Conceitos básicos. Noções de configuração. Administração de serviços de diretório: Active Directory e LDAP. Administração de usuários, grupos, permissões, controles de acesso. Administração de serviços de rede: DNS, DHCP, impressão e compartilhamento de arquivo. Remote desktop. VDI. PowerShell. Interoperabilidade.
Aula 09	Linux (Parte 1) - Conceitos Básicos: Sistema Multitarefa e Multiusuário, Execução em Foreground ou Background, Agendamento de Tarefas, Sistemas de Arquivos, LVM (Logical Volume Manager), Estrutura dos Diretórios, Memória Virtual, Gerenciadores de Pacotes, Configuração de Serviços ou Funcionalidades, Interoperabilidade (Samba), Serviço de Diretório. Shell Script.
Aula 10	Linux (Parte 2) - Serviços de Rede (DNS, DHCP, FTP, E-mail). Gerenciamento de Processos. Montagem de Volumes. Superusuário. Usuários, Grupos, Permissões de Acesso. Comandos Básicos.

Aula 11	Virtualização: Conceitos. Tipos de hypervisor. Sistemas de virtualização: VMware, Hyper-V, Xen Server, Linux Container, Docker.
Aula 12	Computação em Nuvem: Conceitos. PaaS, IaaS, SaaS. Nuvem Pública. Nuvem Privada. Nuvem Híbrida. Soluções. Backup: conceitos, políticas de backup, tipos.
Aula 13	Padrões de Discos e Interfaces. Redes de Armazenamento: conceitos e tipos. RAID.
Aula 14	Ambiente Datacenter: Conceitos. Arquiteturas. Tipos. Normas. Subsistemas principais de Data Center. Replicação. Fail-over. Redes de armazenamento.
Aula 15	Sistemas Distribuídos: conceitos, tipos, arquiteturas, protocolos. Servidores de aplicação JEE.
Aula 16	Servidores Web e proxy: conceitos, funcionamento e configuração.

Essa é a distribuição dos assuntos ao longo do curso. Eventuais ajustes poderão ocorrer, especialmente por questões didáticas.

INTRODUÇÃO AO ESTUDO DE SISTEMAS OPERACIONAIS E ARQUITETURA DE COMPUTADORES

Considerações Iniciais

Na aula de hoje vamos estudar assuntos iniciais, que vão desde a diferença entre organização e arquitetura de computadores, as arquiteturas clássicas de computadores e as arquiteturas clássicas de processadores. Na sequência veremos o que são os conjuntos e os ciclos de instruções, e começaremos a "subir" o nível, ou seja, veremos onde "entra" o software, os algoritmos e linguagens de programação e as linguagens de máquina e de montagem. Por fim, veremos as diferenças entre tradução, compilação e interpretação. Boa aula!

Arquitetura e Organização de Computadores

Processamento de Dados, Organização e Arquitetura

Um computador é uma máquina capaz de coletar, manipular e dar resultados da manipulação de informações. Por ter essas características, o computador já foi chamado de equipamento de processamento eletrônico de dados.

A manipulação das informações coletadas é chamada de **processamento** e as informações iniciais são chamadas **dados**, por isso é comum vermos a expressão **processamento de dados**. Dados e informações podem ser considerados sinônimos, mas quando tratados como distintos, **dado** quer dizer a matéria-prima coletada em uma ou mais fontes (ex.: valores coletados de um teclado), e **informação** significa o resultado do processamento, ou seja, o dado processado.

Quando se estuda um computador, há dois pontos de vista a serem analisados: o da organização (ou implementação) e o da arquitetura de um computador.

A **organização de um computador** é a parte do estudo da ciência da computação que trata dos aspectos relativos à parte do computador mais conhecida por quem o construiu (**detalhes físicos**). Tais entendimentos são desnecessários ao programador que já recebe a máquina pronta, entende a linguagem de programação a ser realizada e utiliza um compilador ou montador para gerar o executável. Alguns exemplos dos aspectos relativos aos componentes físicos são:

- Tecnologia utilizada na construção da memória;
- Frequência do relógio;
- Sinais de controle para iniciar as micro-operações em diversas unidades do computador.

A **arquitetura de um computador** atua mais no nível de conhecimento interessante ao **programador**, pois suas características possuem impacto direto no desenvolvimento de um programa. Alguns exemplos são:

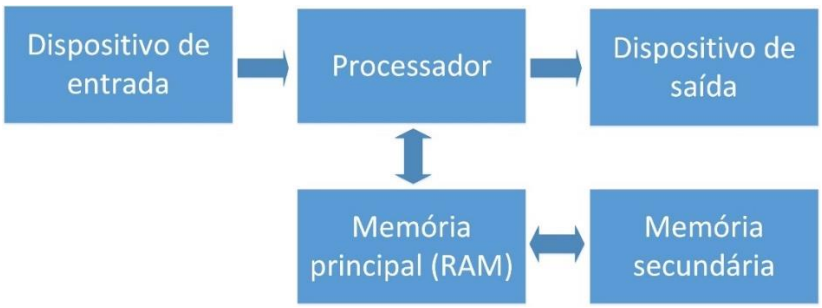
- Conjunto de instruções do processador (ex.: ADD, SUB, entre outras);
- Tamanho da palavra (quantidade de bits utilizada para transferência entre o processador e a memória - ex.: palavra de 32 bits);
- Modos de endereçamento das instruções (relativo, indexado, entre outros);
- Tipo dos dados manipulados.

Para deixar mais claro, vamos falar da "família" de processadores x86. A Intel (fabricante) definiu elementos característicos dessa **arquitetura** (x86), sendo que cada modelo de processador possui sua **organização**. Dessa forma, se um programa foi feito para ser executado em um antigo 80386 (fui longe agora, né? 😊), o mesmo pode ser executado em processadores sucessores (80486, Pentium e posteriores), sem precisar de alterações! Isso ocorre porque são processadores da mesma "família", logo possuem a mesma arquitetura (e isso interessa aos programadores!).

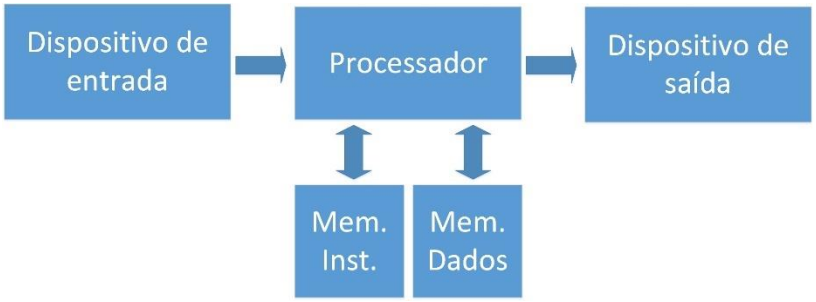
Arquiteturas Clássicas

Um **sistema de computação** é um conjunto de componentes que são integrados para funcionar como se fosse um único elemento, tendo como objetivo realizar o processamento de dados e obter resultados. Os primeiros computadores surgiram com dispositivos de entrada (ex.: teclado), processador (também conhecido como CPU – Unidade Central de Processamento) e dispositivos de saída (ex.: monitor de vídeo).

Em seguida, John von Neumann melhorou a arquitetura inicial, acrescentando a memória (principal e secundária) para armazenar programas e dados, tornando o processamento muito mais rápido e eficaz. Tal arquitetura, embora tenha tido ajuda de outras pessoas, recebeu o nome de **Arquitetura de von Neumann** (figura abaixo). Essa arquitetura tem se mantido ao longo do tempo, com um grande aumento de velocidade (Obs.: a memória secundária não costuma aparecer em figuras da Arquitetura de von Neumann, geralmente aparece apenas "Memória" de forma genérica).



Um melhoramento da Arquitetura de von Neumann é a **Arquitetura de Harvard**, tendo surgido da necessidade de colocar o microcontrolador para trabalhar mais rápido. É uma arquitetura de computador que se distingue das outras por possuir **duas memórias diferentes e independentes em termos de barramento e ligação ao processador**. Sua principal característica é o acesso à memória de dados de modo separado em relação à memória de instruções (programa), o que é tipicamente adotado pelas memórias cache na atualidade:



Com essa separação de dados e instruções em memórias e barramentos separados, o processador consegue acessar as duas simultaneamente, obtendo um desempenho melhor do que o da Arquitetura de von Neumann, pois pode buscar uma nova instrução enquanto executa outra.

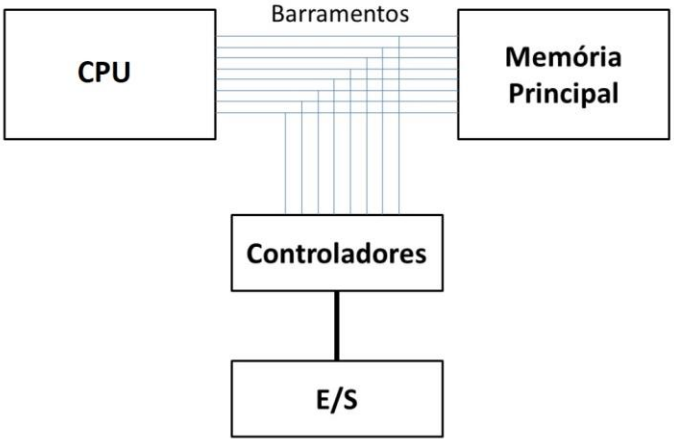
Obviamente que podem existir outras arquiteturas, mas essas duas são as mais utilizadas até hoje e, o mais importante, são cobradas em provas de concurso!

De uma forma geral, são funções básicas de um computador:

- **Processamento** de dados: realizado pelo **processador** (CPU – Unidade Central de Processamento);
- **Armazenamento** de dados: pode ocorrer de forma temporária (dados em uso durante o processamento: **memória principal**) ou de longo prazo (**memória secundária** ou mídias de armazenamento, ex.: HD);
- **Transferência** de dados: ocorre através de **sistemas de interconexão (barramento do sistema)**, permitindo a comunicação com dispositivos de entrada e saída (diretamente conectados ao computador) ou a comunicação de dados a um dispositivo remoto (através de redes de computadores);
- **Controle**: uma unidade de controle **gerencia os recursos do computador**.

Como podemos ver na figura ao lado, um computador na atualidade continua utilizando a essência da Arquitetura de von Neumann e/ou a Arquitetura de Harvard.

Os barramentos são os responsáveis pela comunicação entre o processador, a memória principal e os dispositivos de entrada (teclado, mouse, caneta ótica etc.), saída (monitor, impressora etc.) e os híbridos (dispositivos de armazenamento como cartão de memória, pen drive, HD etc.).



Questões Comentadas

1. **(FUNIVERSA/IPHAN - 2009) Um sistema de processamento de dados é composto, basicamente, por três etapas: (1) entrada de dados, (2) processamento ou tratamento da informação e (3) saída. Em um computador, essas tarefas são realizadas por partes diversas que o compõem, como teclado, mouse, microprocessador, memória etc. Levando-se em conta as tarefas de processamento de dados realizadas por um computador, é correto afirmar que**

A) dispositivos de hardware como teclado e mouse são responsáveis pela saída de dados, uma vez que escrevem ou apontam o resultado esperado em uma operação realizada pelo computador.

B) acessórios modernos como webcams, bluetooth e leitores biométricos são dispositivos de saída de dados incorporados a alguns computadores como acessórios de fábrica.

C) a tela (ou monitor) de um computador comporta-se como um dispositivo de entrada de dados, quando se trabalha em sistemas de janelas, com botões a serem “clicados” pelo usuário.

D) as impressoras multifuncionais são dispositivos mistos, de entrada, processamento e saída de dados, pois podem ler (scanner), processar (memória interna) e imprimir informações.

E) a entrada de dados é tarefa realizada pela pessoa (ou por um programa de computador) responsável por alimentar o sistema com dados necessários para atingir o resultado esperado.

Comentários:

(A) Teclado e mouse são dispositivos de entrada de dados (do ponto de vista do computador, recebem dados); (B) Webcams e leitores biométricos também são dispositivos de entrada e bluetooh é um padrão de rede sem fio com curta distância; (C) O monitor é um dispositivo de saída, pois mostra dados (imagem) e não recebe; (D) São dispositivos de E/S (a função de scanner é de entrada, a função de impressora é de saída), não há processamento em memória interna – processamento é realizado por processador! **(E) A entrada de dados pode ser realizada por uma pessoa, através de um dispositivo de entrada (ex.: teclado). Esses dados alimentam o sistema, que são processados e resultados são gerados (mostrados no monitor, por exemplo).**

2. **(MS CONCURSOS/CODENI-RJ - 2010) É o componente vital do sistema, porque, além de efetivamente realizar as ações finais, interpreta o tipo e o modo de execução de uma instrução, bem como controla quando e o que deve ser realizado pelos demais componentes, emitindo para isso sinais apropriados de controle. A descrição acima refere-se a?**

A) Dispositivos de Entrada e Saída.

B) Memória Principal.

C) Memória Secundária.

D) Unidade Central de Processamento.

Comentários:

"Quem" realiza o processamento dos dados, bem como o devido controle dos dados a serem carregados em memória, buscados para o processador, entre outras atividades, é o processador (também conhecido por CPU – Unidade Central de Processamento). Portanto, a **alternativa D** está correta e é o gabarito da questão.

3. (CESPE/EBC - 2011) São funções básicas de um computador: processamento de dados, armazenamento de dados, transferência de dados e controle. São componentes estruturais de um computador: unidade central de processamento, memória principal, dispositivos de entrada e saída e sistemas de interconexão.

Comentários:

Um computador processa dados (através da CPU), armazena (através de memórias primárias e secundárias) e transfere (através de barramentos, ou sistemas de interconexão) tanto para componentes internos como para dispositivos de entrada (teclado, mouse etc.) e saída (impressora, monitor etc.). Portanto, a questão está **correta**.

4. (AOCP/TCE-PA - 2012) Em computação CPU significa

A) Central de Processamento Única.

B) Único Centro de Processamento.

C) Unidade Central de Processamento.

D) Central da Unidade de Processamento.

E) Centro da Unidade de Processamento.

Comentários:

CPU = Central Processing Unit (Unidade Central de Processamento). Portanto, a **alternativa C** está correta e é o gabarito da questão.

5. (MS CONCURSOS/IF-AC - 2014) Dentre as funções básicas do computador, podemos citar, exceto:

A) Entrada de dados.

B) Processamento de Dados.

C) Saída de Informações.

D) Capacidade de Unidade.

Comentários:

A figura mais básica sobre as funções básicas de um computador:

Portanto, a **alternativa D** está correta e é o gabarito da questão.

6. (CESPE/Polícia Científica-PE - 2016) Assinale a opção correta acerca da arquitetura Harvard de microprocessadores.

A) É a arquitetura mais antiga em termos de uso em larga escala

B) Não permite pipelining.

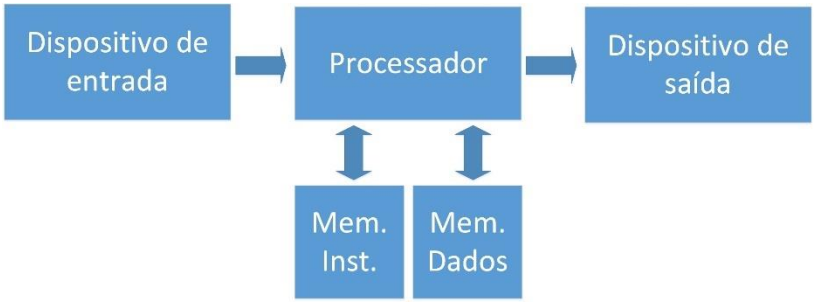
C) Não permite o uso de um conjunto reduzido de instruções.

D) Dispensa a unidade lógica aritmética

E) Apresenta memórias de programa e de dados distintas e independentes em termos de barramentos.

Comentários:

Um melhoramento da Arquitetura de von Neumann é a Arquitetura de Harvard, tendo surgido da necessidade de colocar o microcontrolador para trabalhar mais rápido. É uma arquitetura de computador que se distingue das outras por possuir duas memórias diferentes e independentes em termos de barramento e ligação ao processador. Sua principal característica é o acesso à memória de dados de modo separado em relação à memória de instruções (programa), o que é tipicamente adotado pelas memórias cache na atualidade:



Portanto, a **alternativa E está correta e é o gabarito da questão.**

7. (UFMT/UFSBA - 2017) A respeito de memória cache, os projetos denominados arquitetura Harvard são aqueles

A) cuja cache é unificada, com dados e instruções na mesma cache.

B) cujos conceitos do princípio da localidade foram descartados e adotou-se um protocolo serial de acesso a dados.

C) cuja cache é dividida, com instruções em uma e os dados em outra.

D) cujo empacotamento de módulos de memória cache foi colocado fora do chip, reduzindo o custo de produção e aumentando a quantidade de memória disponível.

Comentários:

Acabamos de ver na questão anterior 😊. Portanto, a **alternativa C está correta e é o gabarito da questão.**

8. (UFPA/UFPA - 2017) O gargalo de von Neumann é caracterizado pela maior velocidade de processamento do processador em relação ao que a memória pode servir a ele. Para minimizar esse gargalo, é necessário

A) utilizar sempre as versões mais atualizadas dos sistemas operacionais.

B) utilizar memória cache entre o processador e a memória principal com caminhos separados para dados e instruções.

C) utilizar processadores de 32 bits ao invés de 64 bits.

D) aplicar o processo de desfragmentação do disco.

E) bloquear a utilização de algoritmos e lógicas de branchpredictor.

Comentários:

Como o processador é mais rápido que a memória, uma solução adotada há um bom tempo é o uso de memórias cache, as quais mantêm as instruções e dados mais acessados, evitando ter que buscar da memória RAM (o que seria mais lento). E, para melhorar ainda mais, as memórias cache começaram a separar os dados das instruções, aplicando o conceito da Arquitetura de Harvard (que é um melhoramento da Arquitetura de von Neumann). Portanto, a **alternativa B está correta e é o gabarito da questão**.

9. (INAZ do Pará/CFF - 2017) A arquitetura de computadores de Von Neumann é frequentemente definida como o conjunto de atributos da máquina que um programador deve compreender para que consiga programar o computador específico com sucesso, e também são compostas de três subsistemas básicos. Assinale a alternativa correta que apresenta os três subsistemas básicos.

A) CPU, memória principal e sistema de entrada e saída.

B) Vídeo, memória externa e não volátil e sistema de entrada e saída.

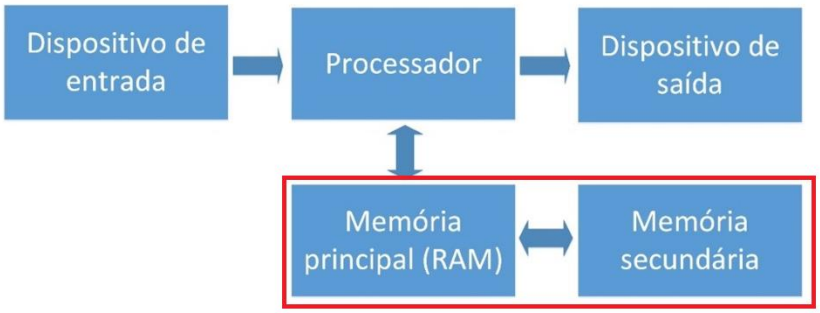
C) CPU, memória secundária e sistema de entrada e saída.

D) CPU, memória principal e sistema operacional.

E) Vídeo, memória secundária e sistema de entrada e saída.

Comentários:

A figura que utilizamos na aula é mais abrangente, mas de uma forma mais simples poderíamos visualizar assim (esquecendo a memória secundária):



Embora seja necessário para a comunicação entre os componentes, os barramentos não são considerados como um subsistema básico, então sobraram os três: processador, memória e dispositivos de E/S. Portanto, a **alternativa A está correta e é o gabarito da questão**.

10. (CESPE/ABIN - 2018) Na arquitetura de Von Neumann, o caminho único de dados é o barramento físico, que liga a memória diretamente aos dispositivos de entrada e saída (E/S): o objetivo desse barramento é a troca de dados externos com a máquina, enquanto a memória guarda os dados de forma temporária no computador.

Comentários:

Podemos ver na figura mostrada na questão anterior que a CPU é o elemento central, então o barramento faz a ligação dela com a memória e dela com os dispositivos de E/S. Portanto, a questão está **errada**.

11. (COPESE-UFT/UFT - 2018) Em 1952 John von Neumann desenvolveu um protótipo de um novo computador de programa armazenado. Esse projeto ficou conhecido como arquitetura de Von Neumann e ainda hoje influencia o projeto de computadores modernos. Os componentes abaixo fazem parte da arquitetura de Von Neumann, EXCETO:

A) Memória Principal.

B) Unidade Lógica e Aritmética (ALU).

C) Barramento.

D) Equipamento de Entrada e Saída (E/S).

Comentários:

Mais uma vez uma questão que deixa o barramento de fora, como se aquelas "caixinhas" se comunicassem através do ar. Mas é assim mesmo, temos que pensar que os 3 componentes da Arquitetura de von Neumann são: processador, memória e dispositivos de E/S. Na questão aparece a ULA (Unidade Lógica e Aritmética), que é um componente de um processador, então consideramos como processador na questão. Portanto, a **alternativa C está correta e é o gabarito da questão**.

12. (INSTITUTO PRÓ-MUNICÍPIO/CRP-11ª Região - 2019) O computador é uma máquina que processa informações eletronicamente, na forma de dados e pode ser programado para as mais diversas tarefas. As fases do processamento são:

A) Monotarefa, Monousuário e Multitarefa;

B) Entrada de dados, Processamento e Saída de Dados;

C) Operação, Linguagem e Aplicação;

D) Programação, Instalação e Registro de Dados.

Comentários:

Mais uma vez aquela figura:

Está certo que a questão fala em fases do "processamento" e o correto seria algo como "funções básicas de um computador", mas tudo bem... a **alternativa B está correta e é o gabarito da questão**.

Arquiteturas de Processadores (RISC e CISC)

Conceitos

Quando o assunto é saber qual a melhor arquitetura de processador, sempre há polêmica. Muitos defendem que os "Macs" são mais rápidos por terem chips RISC, por exemplo. Mas o que é RISC? E CISC? Quais vantagens e desvantagens? Vamos lá...

Um processador **CISC** (*Complex Instruction Set Computer* - Computador com um Conjunto Complexo de Instruções), é capaz de executar **várias centenas de instruções complexas diferentes**, sendo extremamente versátil. Alguns exemplos de processadores CISC são o 386 e o 486.

Alguns fabricantes decidiram seguir o caminho contrário, criando o padrão **RISC** (*Reduced Instruction Set Computer* - Computador com um Conjunto Reduzido de Instruções). Os processadores RISC são capazes de executar apenas **algumas poucas instruções simples**. Justamente por isso, os chips baseados nesta arquitetura são **mais simples e muito mais baratos**.

Outra vantagem dos processadores RISC, é que, por terem um menor número de circuitos internos, podem trabalhar a frequências mais altas. Alguns exemplos de processadores RISC são Sparc (Sun), Mips (Silicon Graphics), Power (IBM) e Alpha (DEC).

Aí surge a dúvida...como um chip que é capaz de executar algumas poucas instruções pode ser considerado por muitos, mais rápido do que outro que executa centenas delas? A grande questão é que um processador RISC é capaz de executar suas poucas instruções muito mais rapidamente. A ideia principal é que apesar de um processador CISC ser capaz de executar centenas de instruções diferentes, apenas algumas são usadas frequentemente, o que parece ser um desperdício, não?

Mas, uma coisa é garantida: em instruções complexas os processadores CISC se saem melhor! O que podemos concluir, então? O ideal é fazer um mix das duas tecnologias, e é por isso que atualmente temos processadores híbridos, que são essencialmente processadores CISC, mas incorporam muitas características dos processadores RISC (ou vice-versa). Tanto os processadores da família x86, como o Pentium II, Pentium III e AMD Athlon, quanto processadores supostamente RISC, como o MIPS R10000 e o HP PA-8000 misturam características das duas arquiteturas, por simples questão de desempenho.

Uma coisa é o mundo real, outra é o mundo dos concursos, onde é importante saber diferenciar bem as características RISC e CISC.

Vamos começar pela **CISC (*Complex Instruction Set Computer*)**:

- Possui grande quantidade de instruções, com múltiplos modos de endereçamento;
- O conceito de microprogramação facilitou o projeto de instruções complexas;
- Microcódigo reside em memória de controle (memória ROM que fica dentro da Unidade de Controle do processador, bem mais rápido que a memória RAM!);
- Criação de novas instruções quase não tem custo (basta ter espaço ainda na memória de controle).

Essas características facilitam a implementação do conceito de famílias de processadores. Por exemplo, na arquitetura x86 houve um acréscimo de instruções do 386 para 486, Pentium, Pentium MMX etc. Ou seja, instruções novas foram introduzidas, aproveitando as antigas, sem ter que começar do zero, sem alterar o projeto básico!

Podemos destacar três aspectos básicos:

- Uso de microcódigo (camada de hardware em nível de instruções ou estruturas de dados envolvidas na implementação do nível superior de código de máquina);
- As instruções são completas e eficientes;
- Instruções de máquina de “alto nível” (complexidade semelhante à dos comandos de alto nível).

Algumas características:

- Formato de 2 operandos é o mais comum, ex.: ADD AX, mem;
- Uso dos modos: Registrador para registrador, Registrador para memória, Memória para registrador;
- Múltiplos modos de endereçamento para a memória, incluindo indexação (vetores);
- Instruções com largura variável;

- Instruções requerem múltiplos ciclos de relógio para completar a execução, ex.: se existe a busca de dois operandos na memória, demora mais;
- Poucos registradores, devido ao pouco espaço no chip (tem memória para o microcódigo, decodificador etc.) e à possibilidade de acesso a operandos na memória;
- Há registradores especializados: controle (*flags*), segmento (ponteiro da pilha) etc.

Agora vamos para a arquitetura **RISC (*Reduced Instruction Set Computer*)**:

- Possui poucas instruções e todas possuem a mesma largura;
- Execução otimizada de chamada de funções;
- Menor quantidade de modos de endereçamento;
- Uso intenso de *pipelining*, pois é mais fácil implementar o paralelismo quando se tem instruções de mesmo tamanho;
- Execução rápida de cada instrução (uma por ciclo de relógio);
- Processadores RISC não requerem microcódigos (sobra mais espaço no chip);
- Menos acesso à memória principal, instruções que acessam a memória: LOAD e STORE (arquitetura registrador – registrador, ou seja, após buscar os dados da memória e colocá-los em registradores, as operações são realizadas);
- Maior quantidade de registradores, justamente pelo explicado no item anterior.

Vamos analisar a tabela abaixo, de diferentes processadores e na sequência vamos classifica-los como RISC ou CISC.

Características	MIPS R4000	RS/6000	VAX11/780	INTEL 486
Quantidade de instruções	94	183	303	235
Modos de endereçamento	1	4	22	11
Largura de instruções (bytes)	4	4	2-57	1-12
Quantidade de registradores de uso geral	32	32	16	8

Olhando pela quantidade de instruções, o que apresenta 94 parece ser RISC (poucas instruções) e o 303 CISC (muitas instruções), mas os outros dois são próximos e fica a dúvida.

Analisando os modos de endereçamento, fica evidente que os dois primeiros são RISC (poucos modos), enquanto os dois últimos possuem bem mais.

Pela largura de instruções fica mais claro ainda que os dois primeiros são RISC, pois possuem uma largura fixa de instruções (4 bytes), enquanto os outros dois possuem instruções de diversos tamanhos (2 a 57 bytes um deles e o outro entre 1 e 12 bytes).

E para arrematar nossa análise, os dois que achamos que são RISC possuem mais registradores (32 cada um deles), enquanto os outros dois processadores possuem menos registradores (um possui 16 e o outro 8).

Conclusão: os processadores MIPS R4000 e RS/6000 possuem arquitetura RISC e os processadores VAX11/780 e INTEL 486 possuem arquitetura CISC. Lembrando...essa é uma classificação conforme as características que predominam, mas na realidade os processadores não possuem características apenas de um tipo de arquitetura e são chamados de híbridos.

Questões Comentadas

13. (ESAF/SUSEP - 2010) Em uma Arquitetura RISC

A) há poucos registradores.

B) há pouco uso da técnica pipelining.

C) as instruções possuem diversos formatos.

D) as instruções são realizadas por microcódigo.

E) as instruções utilizam poucos ciclos de máquina.

Comentários:

Em uma arquitetura RISC existem muitos registradores, há muito uso da técnica pipelining (devido ao tamanho fixo das instruções), as instruções possuem poucos formatos, não são realizadas por microcódigo. E por fim, as instruções utilizam poucos ciclos de máquina (um ciclo, na verdade)! Portanto, a **alternativa E está correta e é o gabarito da questão**.

14. (FCC/TRE-AM - 2010) Numa máquina estruturada multinível, é o nível essencial para as máquinas CISC (Complex Instruction Set Computer), mas que inexiste nas máquinas RISC (Reduced Instruction Set Computer). Trata-se do nível

A) do sistema operacional.

B) de lógica digital.

C) de microprogramação.

D) convencional de máquina.

E) do montador.

Comentários:

A microprogramação é utilizada pela arquitetura CISC, o que consome espaço no chip (na unidade de controle do processador), algo que não existe na arquitetura RISC. Portanto, a **alternativa C está correta e é o gabarito da questão**.

15. (CESPE/Correios - 2011) As instruções CISC são mais simples que as instruções RISC, por isso, os compiladores para máquinas CISC são mais complexos, visto que precisam compensar a simplificação presente nas instruções. Entretanto, se for usado pipeline, a complexidade do compilador CISC é reduzida, pois a arquitetura pipeline evita a necessidade de reordenação inteligente de instruções.

Comentários:

O nome já deixa claro: "Complex Instruction Set Computer", portanto são mais complexas. Os compiladores para máquinas RISC é que são mais complexos, pois devem lidar com instruções simples. Portanto, a questão está **errada**.

16. (VUNESP/UNESP - 2013) Um computador baseado em uma Unidade Central de Processamento do tipo RISC

1

A) não faz uso de pipeline.

B) executa cada instrução em um ciclo de relógio

C) possui instruções de tamanho variável.

D) possui muitos modos de endereçamento

E) possui um grande conjunto de instruções.

Comentários:

Processadores RISC possuem instruções de tamanho fixo e cada instrução é executada em um ciclo de relógio. Portanto, a **alternativa B está correta e é o gabarito da questão**.

17. (FUNDEP/IPSEMG - 2013) A arquitetura RISC de um computador possui as seguintes características, EXCETO:

A) Formatos simples de instruções.

B) Modos simples de endereçamento.

C) Operações memória-para-memória.

D) Uma instrução por ciclo.

Comentários:

RISC é tudo "simples" e uma instrução por ciclo de relógio. Realiza operações registrador-registrador, ou seja, tem que buscar da memória os dados antes (através de LOAD). Portanto, a **alternativa C está correta e é o gabarito da questão**.

18. (CESPE/Antaq - 2014) Atualmente, os fabricantes de computadores têm adotado exclusivamente a arquitetura RISC para o desenvolvimento de chips para processadores, dado o melhor desempenho dessa arquitetura em relação à arquitetura CISC.

Comentários:

Esse "exclusivamente" mata, heim! Para começar os fabricantes têm utilizado uma arquitetura híbrida, com mais características de uma ou de outra. Portanto, a questão está **errada**.

19. (IADES/PCDF - 2016) Em relação ao projeto de máquinas RISC e CISC, assinale a alternativa correta.

A) Dadas as características das instruções das máquinas CISC, o pipeline fica favorecido nessa arquitetura.

B) Arquiteturas RISC normalmente realizam poucas operações de registrador para registrador, aumentando o acesso à memória cache.

1

C) Programas para arquiteturas CISC sempre possuem tamanho menor que programas para arquiteturas RISC, devido à relação um para um de instruções de máquina e instruções de compilador.

D) Arquiteturas RISC tendem a enfatizar referências aos registradores no lugar de referências à memória.

E) Arquiteturas CISC usam um número muito grande de instruções simples em detrimento de instruções complexas.

Comentários:

Processadores da arquitetura só utilizam LOAD e STORE para acessar a memória, depois só realizam operações envolvendo dados que estão nos registradores. Portanto, a **alternativa D está correta e é o gabarito da questão**.

20. (INAZ do Pará/CORE-SP - 2019) "O projeto do Conjunto de Instruções inicia com a escolha de uma entre duas abordagens, a abordagem RISC e a CISC".

Disponível em: http://producao.virtual.ufpb.br/books/edusantana/introducao-a-arquitetura-de-computadores-livro/livro/livro.chunked/ch04s04.html. Acesso em: 13.12.2018.

Quais são características do paradigma RISC de projeto de CPU?

A) São mais baratos, menos acesso à memória, conjunto de instruções simples.

B) Objetivo de criar um hardware mais otimizado, com isso os programas tendem a ocupar menos espaço em memória.

C) Grande número de registradores de propósito geral e os programas tendem a ocupar menos espaço em memória.

D) Em geral usa mais memória para armazenamento de dados.

E) Muitos modos de endereçamento, e foco no hardware.

Comentários:

São mais baratos, pois são mais simples. Ocorrem menos acessos à memória (apenas LOAD e STORE). Possui um conjunto de instruções simples, ao contrário da CISC (C = Complex). Portanto, a **alternativa A está correta e é o gabarito da questão**.

21. (Quadrix/CRA-PR - 2019) Possuir um conjunto de instruções simples e limitado é uma das principais características da arquitetura CISC.

Comentários:

Simples = RISC! "Complexo" = CISC! Portanto, a questão está **errada**.

7

22. (Quadrix/CRA-PR - 2019) A característica que mais se destaca na arquitetura RISC é que computadores pertencentes a ela realizam milhares de instruções por ciclo.

Comentários:

Vamos relembrar as características da a arquitetura RISC (*Reduced Instruction Set Computer*), sendo que destaquei três itens, os quais ajudam a responder esta questão e a seguinte:

- Possui poucas instruções e todas possuem a mesma largura;
- Execução otimizada de chamada de funções;
- **Menor quantidade de modos de endereçamento;**
- Uso intenso de *pipelining*, pois é mais fácil implementar o paralelismo quando se tem instruções de mesmo tamanho;
- **Execução rápida de cada instrução (uma por ciclo de relógio);**
- Processadores RISC não requerem microcódigos (sobra mais espaço no chip);
- **Menos acesso à memória principal, instruções que acessam a memória: LOAD e STORE (arquitetura registrador – registrador, ou seja, após buscar os dados da memória e colocá-los em registradores, as operações são realizadas);**
- Maior quantidade de registradores, justamente pelo explicado no item anterior.

Portanto, a questão está **errada**.

23. (Quadrix/CREA-GO - 2019) Uma máquina RISC, geralmente, usa um conjunto de modos de endereçamento relativamente simples e direto.

Comentários:

Como existem menos modos de endereçamento e eles possuem menos acesso à memória, temos um conjunto de modos de endereçamento mais simples e direto. Portanto, a questão está **correta**.

24. (CESPE/TJ-PA - 2021) Na tentativa de solucionar o chamado espaço semântico (semantic gap), fabricantes de computadores de grande porte criaram alternativas para resolver o problema, como, por exemplo,

I maior densidade de código a ser executado.

II utilização em larga escala do pipelining.

III execução otimizada de chamadas de funções via registradores.

A arquitetura CISC contempla

A) apenas o item I.

B) apenas o item II.

C

C) apenas os itens I e III.

D) apenas os itens II e III.

E) todos os itens.

Comentários:

O tal "espaço semântico" significa a diferença entre os conjuntos de programação de alto nível em várias linguagens de computador e as instruções simples de computação com as quais os microprocessadores trabalham na linguagem de máquina. Na verdade, essa parte só serve para distrair o candidato!

O foco é analisar as características elencadas nos itens I a III e verificar quais delas estão relacionadas à arquiteturas CISC. Vamos lá...

I. Uma maior densidade do código significa que cada instrução deve "fazer muito", de modo que o programa completo tenha poucas instruções - Isso é característica da CISC!

II. Um grande uso de pipeline é característica da RISC!

III. Como a RISC é uma arquitetura registrador-registrador, ela é quem possui uma execução otimizada de chamadas de funções via registradores.

Portanto, a **alternativa A está correta e é o gabarito da questão**.

Hardware x Software

Máquina Multinível

Retomando um pouco sobre a organização de um computador (aspectos relativos à parte do computador mais conhecida por quem o construiu - detalhes físicos), vamos ver o que é uma máquina multinível. Vamos

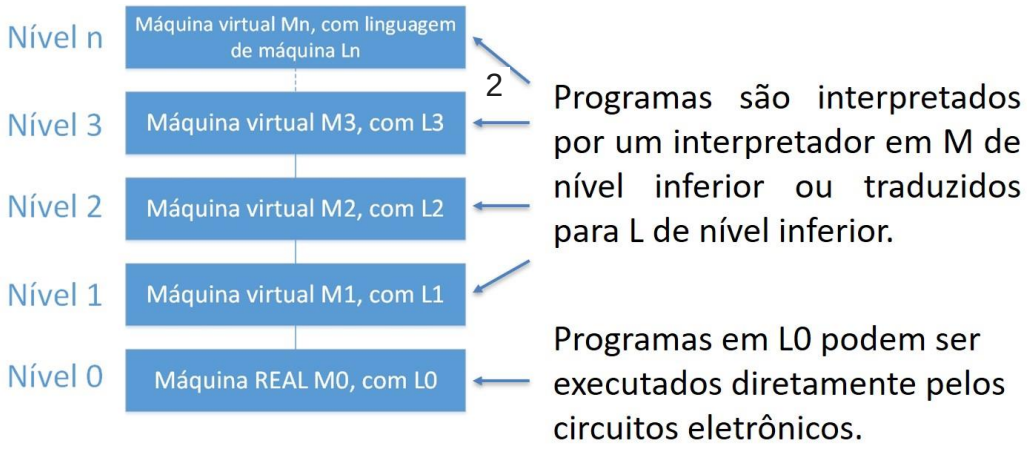
partir do seguinte: as pessoas querem fazer alguma coisa (X), mas os computadores só podem fazer outras coisas (Y). Qual seria a solução? Criar máquinas virtuais em diferentes níveis! Como assim? Veremos...

Basicamente temos duas técnicas:

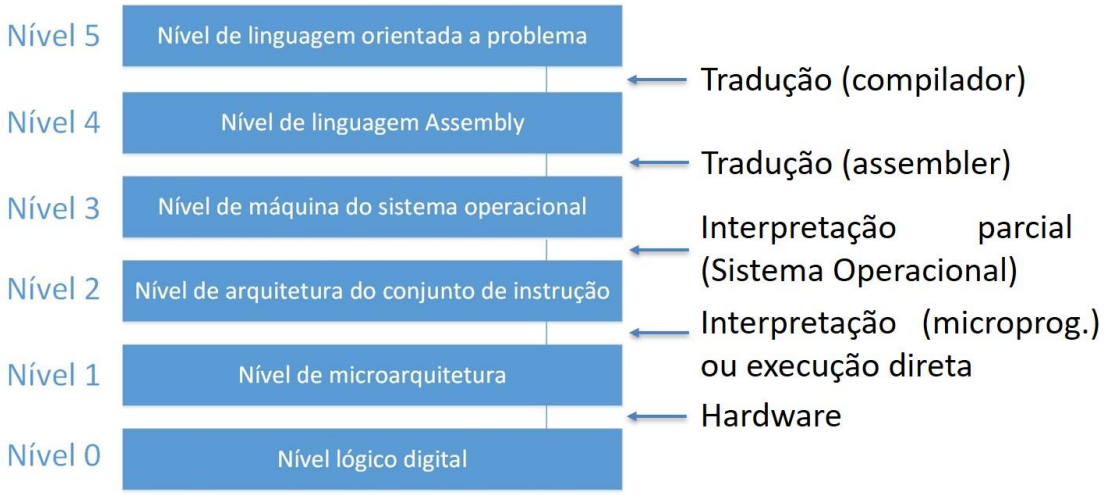
- **Tradução**: Executa um programa escrito em L1 (nível 1), substituindo cada instrução por uma sequência equivalente de instruções em L0 (nível 0);

- **Interpretação**: Escreve-se um programa em L0 que considere os programas em L1 como dados de entrada e os executa, uma instrução por vez, sem criar um novo programa em L0.

Abaixo podemos ver uma estrutura em níveis de uma forma genérica.



Abaixo podemos ver uma estrutura em níveis de máquinas contemporâneas e na sequência uma breve explicação de cada nível.



- Níveis 4 e 5: voltados para os programadores de aplicações (um problema a solucionar);
- Nível 3: conjunto de novas instruções, uma organização de memória diferente, com capacidade de executar dois ou mais programas, entre outras características;
- Nível 2: os fabricantes publicam um manual de referência da linguagem de máquina;
- Nível 1: ULA (Unidade Lógica e Aritmética) + registadores;

- Nível 0: hardware "verdadeiro", circuitos executam os programas em linguagem de máquina do nível 1.

As linguagens de máquina nos níveis 1, 2 e 3 são numéricas e as linguagens nos níveis 4 e 5 possuem palavras e abreviações.

Algoritmos e Linguagens de Programação de Alto Nível

Vamos começar "de cima", ou seja, do ser humano, e vamos aos poucos "baixando o nível" (no bom sentido, é claro), até chegarmos na máquina! Então vamos lá...sabemos que o computador serve para resolver problemas, desde cálculos, edição de textos até jogos de entretenimento, por exemplo. Antes de colocar a mão na massa (programar em alguma linguagem), é importante definir o passo a passo de como resolver o tal problema. Comecemos pelo conceito de **algoritmo**, a seguir.

Na matemática, um algoritmo pode ser definido como uma sequência finita de regras, raciocínios ou operações que, aplicada a um número finito de dados, permite solucionar classes semelhantes de problemas.

Na informática, um algoritmo pode ser definido como um **conjunto das regras e procedimentos lógicos** perfeitamente definidos que levam à **solução de um problema em um número finito de etapas**.

Um exemplo clássico de algoritmo não-computacional é uma receita de bolo, pois você tem todos os ingredientes e a sequência de passos para resolver o "problema" (preparar o bolo). Para fazer um bolo de laranja pode existir uma infinidade de receitas (algoritmos), tendo como resultado um bolo de laranja! Claro que um pode ficar mais ou menos saboroso, dependendo do paladar de cada um!

Todas as tarefas executadas pelo computador são baseadas em algoritmos. Um algoritmo deve ser bem definido, pois é uma máquina que o executará, não haverá "alguém" para decidir o que fazer quando houver alguma ação ambígua a ser executada! Uma calculadora por exemplo, para executar a operação de multiplicação, executa um algoritmo que calcula somas até um determinado número de vezes, porque a operação de multiplicação geralmente não está presente nas instruções conhecidas pela arquitetura (hardware).

As variáveis são os endereços de dados salvos na memória do computador e usados na realização de cálculos pelos algoritmos. Funciona como os ingredientes da receita. O programa é a representação do algoritmo, ou seja, um texto escrito em uma linguagem de programação. Funciona como o texto da receita. Resumindo: o algoritmo é algo mais alto nível, pode ser escrito em português estruturado (Portugol) ou algo semelhante (Ler, Escrever, entre outros comandos), enquanto uma linguagem de programação tem regras bem definidas quanto à sintaxe de comandos, estruturas de dados aceitas etc.

Vamos ver alguns exemplos do Portugol e suas explicações abaixo.

inicio
  texto nome_pessoa
  escrever "Qual o seu nome?"
  ler nome_pessoa
  escrever "Seja bem vindo, ", nome_pessoa
fim
	Foi declarada uma variável de texto (string), denominada nome_pessoa. Escreveu a mensagem "Qual o seu nome?". Leu alguma coisa digitada pelo usuário e armazenou na variável nome_pessoa. Escreveu "Seja bem vindo, " e o nome digitado.

No exemplo acima, a execução no monitor seria assim (em vermelho o que você teria digitado):

Qual o seu nome? **Concurseiro(a) Aprovado(a)**

Seja bem vindo, Concurseiro(a) Aprovado(a)

inicio
  inteiro numero, r
  escrever "Digite um numero: "
  ler numero
  r ← numero % 2
  se r = 0 entao
  escrever "Numero par!"
  senao
  escrever "Numero impar!"
  fimse
fim
	Foram declaradas duas variáveis do tipo inteiro: numero, r. Escreveu a mensagem "Digite um numero:".
	
	Leu algo digitado pelo usuário e armazenou em numero. A variável r recebeu o resultado da divisão inteira de numero por 2.
	
	Se r for igual a 0, então escreve na tela "Numero par!", caso contrário escreve "Numero impar!".

No exemplo acima, a execução no monitor seria assim (em vermelho o que você teria digitado):

Digite um numero: **9**

Numero impar!

Existem algumas **classificações quanto à maneira pelos quais foram implementados**, mas duas delas merecem destaque, os **algoritmos recursivos e os iterativos**.

Uma forma bem resumida de explicar é dizer que **recursão é autorreferência**. Recursão ocorre quando algo é definido em termos de si mesmo ou de suas variações, ou seja, recursão envolve estruturas aninhadas. Uma figura que retrata a recursão é a seguinte (criada por Paul Noth, para o The New Yorker):

Para entender o conceito de recurso, vamos a um exemplo clássico, o cálculo do fatorial de um número. O fatorial de um número n, denotado por n!, é calculado como a seguir:

$$n! = n \times (n-1)!$$

A definição acima é incompleta porque não dos diz como calcular o fatorial de nenhum número específico. Se tentarmos aplicá-la ao cálculo do fatorial de algum número, nunca seremos capazes de parar. Continuaremos calculando indefinidamente.

Agora vamos para uma definição completa: O fatorial de um número n, denotado por n!, é calculado como a seguir:

$$0! = 1$$

$$n! = n \times (n-1)!$$

Com a adição de mais uma regra, sabemos que, ao chegar no zero, conseguiremos calcular o fatorial e poderemos usar esse resultado no cálculo do fatorial dos demais números, até chegarmos ao fatorial do número desejado inicialmente. O procedimento não executará indefinidamente, ou seja, temos uma "condição de parada", uma regra que define até onde calculamos.

Para calcular o fatorial de 10, por exemplo, basta saber como calcular o fatorial de 9, pegar esse resultado, e multiplicá-lo por 10. Só isso! E para calcular o fatorial de 9? Basta calcular o fatorial de 8 e multiplicar o resultado por 9. E assim por diante... até chegar no cálculo do fatorial de 0, que sabemos que é 1. Agora, é só pegar esse resultado e ir multiplicando… até chegar ao ponto em que consegue calcular o fatorial de 10.

Vamos ver o cálculo do fatorial de 5 completo, abaixo:

fatorial(5) = 5 × fatorial(4) =

5 × (4 × fatorial(3)) =

5 × (4 × (3 × fatorial(2))) =

5 × (4 × (3 × (2 × fatorial(1)))) =

5 × (4 × (3 × (2 × (1 × fatorial(0))))) =

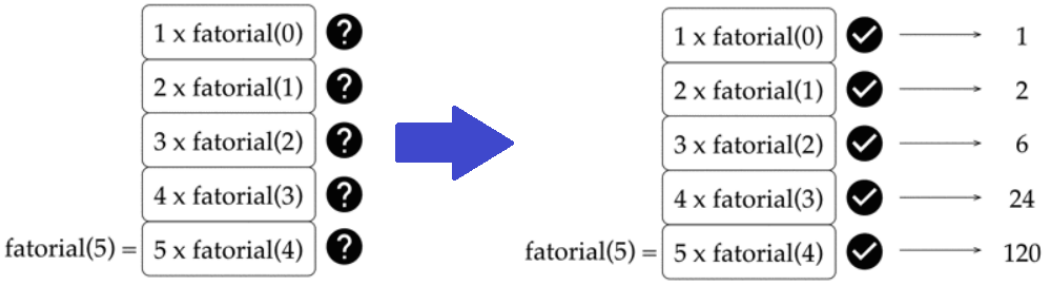
5 × (4 × (3 × (2 × (1 × 1)))) =

5 × (4 × (3 × (2 × 1))) =

5 × (4 × (3 × 2)) =

5 × (4 × 6) = 5 × 24 = 120

Note que temos uma chamada recursiva de uma "função" fatorial, conforme podemos ver na figura abaixo.



Um benefício de conceitos recursivos é que, em geral, a implementação da definição do conceito em uma linguagem de programação é uma tarefa relativamente simples. Por exemplo, veja abaixo a implementação do cálculo do fatorial de um número. Mesmo não entendendo alguns detalhes do código (em Python), é possível verificar a correspondência entre a definição recursiva do fatorial de um número e a implementação recursiva desta definição.

```
def fatorial(n):

    # Caso base: O fatorial de 0 é 1.

    if n == 0:

        return 1

    # O fatorial de um número n é: n * fatorial(n - 1).

    return n * fatorial(n - 1)
```

No código acima, foi fácil traduzir para uma linguagem de programação as duas partes da definição recursiva do fatorial de um número. É possível ler o código e identificar facilmente o que ele está fazendo.

Outra vantagem de algoritmos recursivos é que eles são, em geral, fáceis de serem lidos. A correspondência entre a definição e a implementação é óbvia. Por outro lado, algoritmos recursivos não são muito eficientes em alguns casos, pois pode ser que eles realizem grandes quantidades de trabalho repetido.

Agora vamos focar nos **algoritmos iterativos**. Vimos há pouco que o fatorial de um número inteiro positivo n nada mais é do que o produto de todos os inteiros positivos menores ou iguais a n. Por exemplo:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Comparando este exemplo com o exemplo do fatorial recursivo, vemos que a principal diferença entre o procedimento recursivo e o iterativo é que o procedimento iterativo contém instruções completas de como realizar uma tarefa, ao passo que o procedimento recursivo contém instruções de como realizar uma tarefa tendo como base instâncias menores da mesma tarefa. Um exemplo tornará essa explicação mais clara.

Relembrando, a definição recursiva do fatorial de um número é a seguinte: O fatorial de um número n, denotado por n!, é calculado como a seguir:

$$0!=1$$

$$n!=n\times(n-1)!$$

A definição iterativa do fatorial é um número é a seguinte: O fatorial de um número n, denotado por n!, é calculado como a seguir:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$$

Como dissemos acima, o procedimento iterativo nos diz cada passo do cálculo do fatorial de um número, enquanto o procedimento recursivo nos diz como calcular o fatorial de um número n baseado no fatorial de um número menor, (n−1, neste caso).

Na prática, é muito fácil distinguir um algoritmo recursivo de um algoritmo iterativo. O **algoritmo recursivo sempre terá uma chamada a si mesmo. O algoritmo iterativo nunca terá isso**. Veja abaixo o algoritmo recursivo para o cálculo do fatorial de um número.

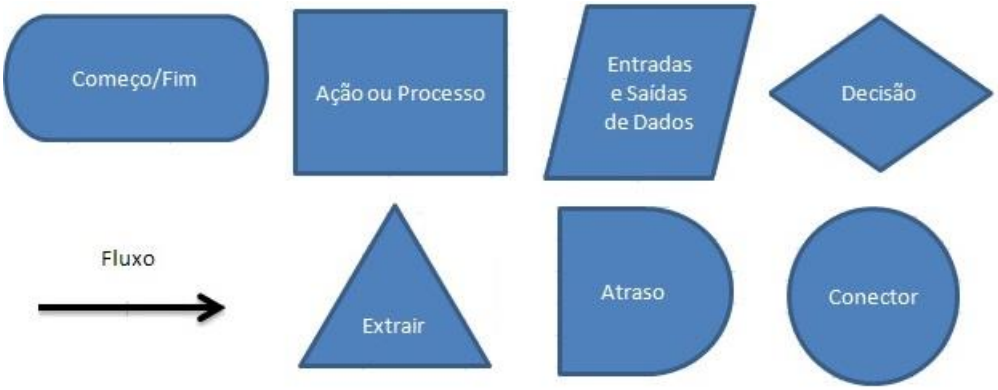
```
def fatorial_iterativo(n):
    resultado_fatorial = 1
    for i in range(1, n + 1):
        resultado_fatorial = resultado_fatorial * i
    return resultado_fatorial
```

No código acima, a chamada à função range(1, n + 1) gera uma lista de números no intervalo [1,n], ou seja, indo de 1 até n. Uma outra forma de calcular o fatorial com um algoritmo iterativo:

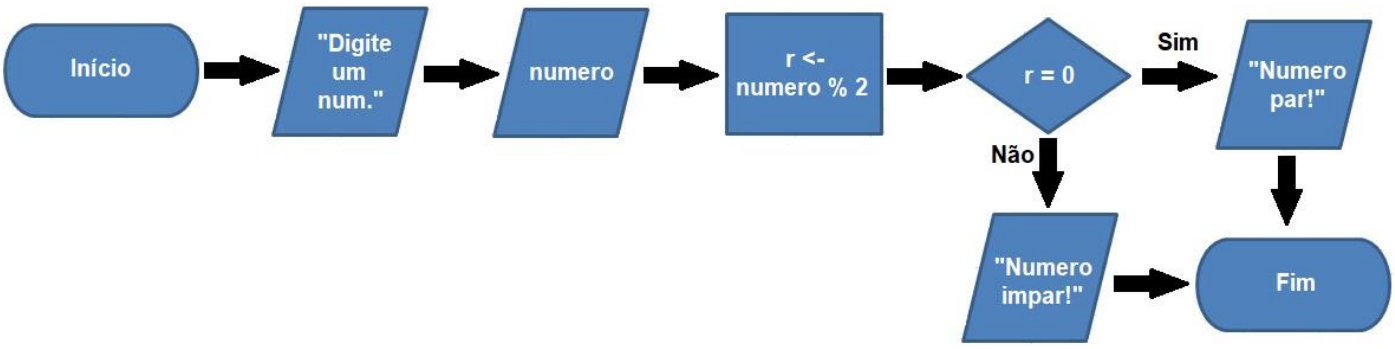
```
def fatorial_iterativo (n):
  resultado_fatorial = 1
  while (n):
    resultado_fatorial *= n
    n -= 1
  return resultado_fatorial
```

Note que nesses dois exemplos não há chamada à "fatorial_iterativo" em nenhum momento.

Outra forma de representar um algoritmo é através de um fluxograma. Se buscarmos o conceito de **fluxograma**, teremos o seguinte: "diagrama para representação de um algoritmo". A maioria dos fluxogramas utiliza até cinco tipos de símbolos, mas existem oito tipos básicos. Eles são conectados um a um por setas, de acordo com o fluxo do processo. Vamos ver os símbolos e seus significados (existem outros conjuntos de símbolos, mas vamos ver apenas esse para ter uma ideia):



Dos símbolos mostrados acima, os cinco primeiros são os mais utilizados. Vamos colocar o nosso segundo algoritmo (aquele que mostra se é número par ou ímpar) em um fluxograma para entender como funciona (vamos assumir que não é necessário declarar as variáveis previamente):



Depois de pronto o algoritmo e/ou o fluxograma é hora de programar! Claro que é possível programar sem ter que fazer qualquer algoritmo antes, mas não é o recomendado. É possível programar em Assembly direto? Claro que sim! Mas é muito complexo e demorado! Por isso, o mais comum é programar em alguma linguagem de alto nível e deixar que o compilador faça o trabalho de transformar o código de alto nível para o executável (se considerarmos aquele compilador que faz tudo, incluindo também o papel de montador e ligador).

A **linguagem de programação** é um método padronizado para comunicar instruções para um computador. É um conjunto de regras sintáticas e semânticas utilizadas para definir um programa (software). Ela permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como tais dados serão armazenados ou transmitidos e quais ações devem ser tomadas em diversas circunstâncias. Resumindo: linguagens de programação são utilizadas para expressar algoritmos com precisão.

O conjunto de palavras (lexemas classificados em tokens), compostos de acordo com essas regras, constituem o código fonte de um software. Tal código é traduzido para código de máquina, o qual é executado pelo processador.

A ideia principal das **linguagens de programação** é que programadores tenham uma maior produtividade, permitindo **expressar seus algoritmos mais facilmente** do que quando comparado com o código de máquina. Por isso, linguagens de programação são projetadas para adotar uma sintaxe de nível mais alto (mais próxima do ser humano).

Outro fator importante é que as linguagens de programação tornam os programas menos dependentes de computadores ou ambientes computacionais específicos, o que é denominado **portabilidade**. Isso ocorre porque programas escritos em linguagens de programação são traduzidos para o código de máquina do computador no qual será executado em vez de ser diretamente executado (no caso de programas compilados). Na sequência vamos ver algumas classificações.

Programação estruturada: é uma forma de programação que preconiza que todos os programas possíveis podem ser reduzidos a apenas três estruturas: sequência, decisão e repetição. A programação estruturada orienta os programadores para a criação de estruturas simples em seus programas, usando as sub-rotinas e as funções.

Programação modular: é uma forma de programação no qual o desenvolvimento das rotinas de programação é feito através de módulos, que são interligados através de uma interface comum.

Programação orientada a objetos (POO): é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos. O extensivo uso de objetos, particularmente em conjunção com o mecanismo de herança, caracteriza o estilo de programação orientada a objetos.

Programação linear: em matemática, problemas de programação linear são problemas de otimização nos quais a função objetivo e as restrições são todas lineares. Esse tipo é mais complexo e dificilmente será cobrado em concurso.

Diferentes linguagens de programação podem ser agrupadas segundo o **paradigma** que seguem para abordar a sua sintaxe e semântica. Os paradigmas se dividem em **dois grandes grupos: imperativo e declarativo**. Os paradigmas imperativos são aqueles que facilitam a computação por meio de mudanças de estado e são divididos em:

- **Paradigma procedural**: os programas são executados através de chamadas sucessivas a procedimentos separados. Exemplos: Fortran e BASIC;
- **Paradigma de estruturas de blocos**: possui como característica principal os escopos aninhados. Exemplos: Pascal e C;
- **Paradigma de orientação a objetos**: descreve linguagens que suportam a interação entre objetos. Exemplos: C++, Java e Python;
- **Paradigma da computação distribuída**: suporta que mais de uma rotina possa ser executada independentemente. Exemplo: Ada.

Os paradigmas declarativos são aqueles nos quais um programa especifica uma relação ou função. Dividem-se em:

- **Paradigma funcional**: linguagens que não incluem qualquer provisão para atribuição ou dados mutáveis. O mapeamento entre os valores de entrada e saída são alcançados mais diretamente. Um programa é uma função (ou grupo de funções), tipicamente constituída de outras funções mais simples. Exemplo: Lisp;

- **Paradigma da programação lógica**: baseia-se na noção de que um programa implementa uma relação ao invés de um mapeamento. Exemplo: Prolog.

Quanto a estrutura de tipos, as linguagens de programação podem ser definidas de duas formas:

- **Fracamente tipada**: quando o tipo da variável muda dinamicamente conforme a situação, ou seja, uma variável pode receber um valor do tipo inteiro, depois string, e por aí vai. Exemplo: PHP;

- **Fortemente tipada**: quando o tipo da variável se mantém o mesmo até ser descartada da memória, ou seja, se uma variável foi declarada como inteiro, ela não pode receber outro tipo de dado. Exemplos: Java e Python.

Em relação a ser dinâmica ou estaticamente tipada:

- **Dinamicamente tipada**: o tipo da variável é definido **em tempo de execução**. Exemplos: Perl, Python e Ruby;

- **Estaticamente tipada**: o tipo da variável é definido **em tempo de compilação**. Exemplos: Java e C.

Teste de Mesa

Teste de mesa é uma **simulação da execução de um programa de forma manual**, geralmente feita no papel (geralmente em cima de uma mesa, por isso o nome!). Não há regras rígidas para criar um teste de mesa, mas geralmente ele é feito do jeito que veremos na sequência. Considere o pseudocódigo abaixo:

```
inteiro a, b, soma, diferenca

escreva("Digite dois números inteiros\n")

leia(a, b)

soma = a + b

diferenca = a - b

escreva("A soma dos dois números é ", soma, "\n")

escreva("A diferença dos dois números é ", diferenca, "\n")
```

Esse algoritmo simplesmente calcula a soma e a diferença entre dois números e imprime ambas. Baseado nele, vamos criar uma tabela onde faremos um teste de mesa. O teste de mesa geralmente é feito de duas formas. Primeiramente, vamos fazer um teste seguindo a primeira forma. Nessa primeira forma, coloca-se as variáveis que se deseja acompanhar o valor em colunas de uma tabela. Então, preenche-se os valores das variáveis para simular a execução do programa. Cada linha corresponde a uma execução.

a	b	soma	diferenca
4	3	7	1
20	10	30	10
10	15	25	-5

Vamos ver agora a segunda forma. Ela também tem uma coluna para cada variável, mas inclui o número da linha na primeira coluna e o valor que cada variável tem após a execução dela. Como cada linha corresponde a uma linha do programa, a tabela inteira corresponde a uma execução. Vamos à tabela:

Linha	a	b	soma	diferenca
4	20	10	30	
5	20	10	30	10

A primeira forma deve ser utilizada quando se deseja simular mais de 1 execução ou quando se acha desnecessário expressar os valores das variáveis em diferentes pontos do algoritmo.

Testes de mesa são mais usados para propósitos didáticos, ou quando não se dispõe de um computador enquanto se está criando um algoritmo e deseja-se testar o algoritmo, geralmente com valores de entrada diferentes. São utilizados também quando se tem dificuldade para entender o funcionamento de um algoritmo. Então, fazendo o teste de mesa, fica mais fácil entender o que o algoritmo faz.

Questões Comentadas

25. (CESPE/TCE-PA - 2016) Utilizando-se linguagens fracamente tipadas, é possível alterar o tipo de dado contido em uma variável durante a execução do programa.

Comentários:

Quanto a estrutura de tipos, as linguagens de programação podem ser definidas de duas formas:

- Fracamente tipada: quando o tipo da variável muda dinamicamente conforme a situação, ou seja, uma variável pode receber um valor do tipo inteiro, depois string, e por aí vai. Exemplo: PHP;
- Fortemente tipada: quando o tipo da variável se mantém o mesmo até ser descartada da memória, ou seja, se uma variável foi declarada como inteiro, ela não pode receber outro tipo de dado. Exemplos: Java e Python.

Logo, a questão está **correta**.

26. (CESPE/TCE-PA - 2016) Acerca de funções e procedimentos em subprogramas, julgue o item que se segue.

```
algoritmo solução1
var
   A, B, X : inteiro
início
   leia (A, B)
     X ← A
     A ← B
     B ← X
   escreva (A, B)
Fim algoritmo.
algoritmo solução2
var
  A, B : inteiro
Procedimento TROCA
  var
     X : inteiro
  início
     X ← A
     A ← B
     B ← X
  fim
início
    leia (A, B)
     TROCA
     escreva (A, B)
Fim algoritmo.
```

No algoritmo solução1 apresentado a seguir as variáveis X, A e B são criadas com escopo global; no algoritmo solução2 apresentado após algoritmo solução1, as variáveis A e B são criadas com escopo global e a variável X com escopo local.

Comentários:

Escopo global é quando uma variável não é declarada dentro de nenhum procedimento, ou seja, ela pode ser utilizada em todo o algoritmo. Escopo local é quando uma variável é declarada dentro de um procedimento e só pode ser utilizada neste procedimento. Logo, a questão está **correta**.

27. (FUNDEP/UFVJM-MG - 2017) Assinale a alternativa que apresenta corretamente a sequência de passos computacionais que transforma a entrada na saída, ou seja, procedimentos necessários para resolver um determinado problema.

A) Algoritmos

B) Arquivos

C) Cases

D) Polinômio

Comentários:

Na matemática, um algoritmo pode ser definido como uma sequência finita de regras, raciocínios ou operações que, aplicada a um número finito de dados, permite solucionar classes semelhantes de problemas.

Na informática, um algoritmo pode ser definido como um conjunto das regras e procedimentos lógicos perfeitamente definidos que levam à solução de um problema em um número finito de etapas.

Logo, a **alternativa A** está correta e é o gabarito da questão.

28. (CESPE/TRE-TO - 2017) Assinale a opção que apresenta o resultado final após a execução do algoritmo precedente.

```
algoritmo
var numero: inteiro
inicio
numero = 12

   se (numero mod 2 = 0) entao
      escreva ("A")
   senao
      escreva ("B")
   fim-se

   se (numero > 12) entao
      escreva ("C")
   fim-se

fim
```

A) B

B) A

C) AC

D) C

E) BC

Comentários:

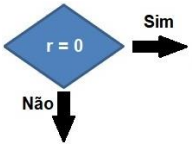
O algoritmo começa com a atribuição do valor 12 à variável numero. A seleção "se (numero mod 2 = 0)" resulta em verdadeiro, é executado escreva("A"). O teste seguinte (numero > 12) resulta em falso, então nada é executado. Portanto, apenas "A" é escrito.

Logo, a **alternativa B** está correta e é o gabarito da questão.

29. (Quadrix/CRM-PR - 2018) Em um fluxograma, as caixas de decisão são como "caixas pretas", uma vez que não se tem clareza da ação que será executada.

Comentários:

Uma caixa de decisão é muito clara, pois tem uma condição e, caso seja verdadeira "vai para um lado", senão "vai para outro":



Logo, a questão está **errada**.

30. (Quadrix/CRM-PR - 2018) Os algoritmos são sequências finitas de instruções que, quando corretamente executadas, levam à solução de um problema.

Comentários:

Os algoritmos são uma sequência de instruções com o objetivo de resolver um problema. Claro que não pode ser infinito, senão não resolveria o problema em vida! Logo, a questão está **correta**.

Linguagens de Máquina e de Montagem

Conceitos

A maioria dos programadores lida com linguagens de alto nível (Pascal, Java, entre outras). Dessa forma, muito pouco da arquitetura da máquina básica fica visível. Um limite onde o projetista de computador e o programador podem ver a mesma máquina é o conjunto de instruções de máquina.

Se um programador deseja programar em linguagem de máquina (na verdade, linguagem de montagem, como veremos em breve), ele deve ter conhecimento da estrutura dos registradores e da memória, dos tipos de dados aceitos diretamente pela máquina e do funcionamento da ULA (Unidade Lógica e Aritmética). Isso tudo porque não há mais um compilador que transforme aquele código de alto nível em um código binário específico para determinada arquitetura!

Os processadores possuem um conjunto de instruções definidas por seus fabricantes e a descrição desse conjunto permite o entendimento sobre o ele (processador). Tendo esse "manual" de instruções aceitas por um determinado processador (ou família de processador), é possível programar em "baixo nível".

Instrução de Máquina

Cada instrução de máquina deve conter as informações exigidas pelo processador para a execução. Os elementos de uma instrução de máquina são:

- **Código de operação**: especifica a operação a ser realizada (ex.: ADD, E/S). A operação é especificada por um código binário conhecido como código da operação (**opcode**);

- **Referência a operando fonte**: a operação pode envolver um ou mais operandos fontes, isto é, operandos que são entradas para a operação;

- **Referência a operando de resultado**: a operação deve produzir um resultado;

- **Referência à próxima instrução**: informa ao processador onde buscar a próxima instrução depois que a execução da instrução atual estiver completa.

Na maioria dos casos, a próxima instrução a ser buscada vem imediatamente após a instrução corrente. Nesses casos, não há uma referência explícita à próxima instrução. Quando necessária uma referência explícita, o endereço da memória principal ou da memória virtual deve ser fornecido.

Os **operandos fonte e resultado** podem estar em uma das quatro áreas mostradas a seguir:

- **Memória principal ou virtual**: assim como as referências à próxima instrução, o endereço da memória (principal ou virtual) deve ser fornecido;

- **Registradores do processador**: um processador tipicamente possui um ou mais registradores, os quais podem ser referenciados por instruções de máquina. Cada registrador recebe um nome ou número exclusivo, e a instrução deve conter o número do registrador desejado;

- **Imediato**: o valor do aperando está contido em um campo na instrução sendo executada;

- **Dispositivo de E/S**: a instrução precisa especificar o módulo e o dispositivo de E/S para a operação. Se a E/S mapeada na memória for utilizada, esse é apenas outro endereço da memória (principal ou virtual).

Representação da Instrução

Cada instrução é representada por uma sequência de bits. A instrução é dividida em campos que correspondem aos elementos que constituem a instrução. Um exemplo de instrução simples de 16 bits é:

4 bits	6 bits	6 bits
Opcode	Referência de operando	Referência de operando

Durante a execução da instrução, uma instrução é lida para um registrador de instrução (IR) no processador. O processador deve ser capaz de extrair os dados dos diversos campos da instrução para realizar a operação exigida.

Como é difícil para o ser humano lidar com representações binárias das instruções de máquina, uma prática comum adotada foi a de utilizar uma representação simbólica das instruções. Um exemplo dessa prática foi utilizado para o conjunto de instruções do IAS[1], o primeiro computador eletrônico construído pelo Instituto de Estudos Avançados de Princeton.

[1] O artigo que descreve o projeto do computador IAS foi editado por nada menos que John von Neumann (nome conhecido, não?), um professor de matemática da Universidade de Princeton e do Instituto de Estudos Avançados.

Os opcodes são representados por abreviações, conhecidas por mnemônicos, os quais indicam a operação. Alguns exemplos comuns são:

- ADD - Adição;
- SUB - Subtração;
- MUL - Multiplicação;
- DIV - Divisão;
- LOAD - Carrega dados da memória;
- STOR - Armazena dados na memória.

Operandos também são representados simbolicamente, como por exemplo a instrução **ADD R, X**, que pode significar a soma do valor contido na localização X com o conteúdo do registrador R.

Modos de Endereçamento

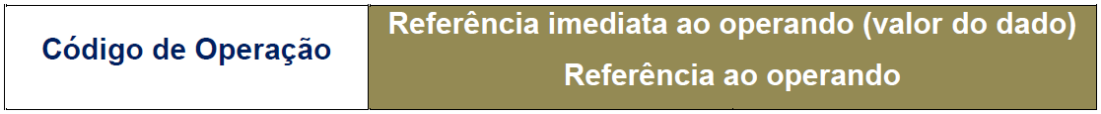
O(s) campo(s) de endereço são relativamente pequenos. Para tornar possível referenciar um grande intervalo de locais da memória principal (ou memória virtual, em alguns sistemas), uma variedade de técnicas de endereçamento foi empregada. Vamos analisar as técnicas ou modos de endereçamento mais comuns. Vamos adotar a seguinte notação:

- A (do inglês, *Address*) = conteúdo de um campo de endereço dentro da instrução;
- R = conteúdo de um campo de endereço dentro da instrução que se refere a um registrador;
- EA (do inglês, *Effective Address*) = endereço real (efetivo) do local que contém o operando referenciado;
- (X) = conteúdos do local de memória X ou do registrador X.

Endereçamento imediato: é a forma mais simples de endereçamento, no qual o valor do operando está presente na instrução:

Operando = VALOR

Esse modo pode ser utilizado para definir e utilizar constantes ou definir valores iniciais das variáveis. A vantagem é que nenhuma referência de memória (além de obter a instrução em si) é necessária para obter o operando. Isso economiza um ciclo de memória ou de cache dentro do ciclo de instrução. A desvantagem é que o tamanho do número é limitado ao tamanho do campo de endereço (geralmente pequeno, se comparado ao tamanho da palavra, que é o tamanho utilizado quando se busca um dado da memória).



Um exemplo que utiliza o modo de endereçamento imediato para mover o valor 20_{16} para o registrador B:
MOV B, #20H

Endereçamento direto: o campo de endereço possui o endereço efetivo do operando:

EA = A

Essa técnica era comum nas primeiras gerações de computadores, requer apenas uma referência à memória e nenhum cálculo especial. A limitação é que ela oferece um espaço de endereçamento limitado.

Endereçamento indireto: no endereçamento direto, o tamanho do campo de endereço geralmente é menor do que o tamanho da palavra, o que limita o espaço de endereços. A solução é ter um campo de endereço fazendo referência ao endereço de uma palavra na memória, a qual possui o endereço completo do operando:

$$EA = (A)$$

Como definido lá nas notações, os parênteses são interpretados como "conteúdo de". A vantagem principal é que, para um tamanho N de uma palavra, um espaço de endereçamento de 2^N ficará disponível. A desvantagem é que a execução da instrução requer duas referências à memória para obter o operando, uma para obter apenas o endereço e a outra para obter o operando (valor) em si.

Endereçamento por registradores: semelhante ao endereçamento direto, porém o campo de endereço faz referência a um registrador em vez de um endereço de memória:

$$EA = R$$

Por exemplo, se o conteúdo de um campo de endereço de registrador for 3, então o registrador R3 é o endereço pretendido e o valor do operando estará em R3. As vantagens são que apenas um pequeno campo de endereço é necessário e nenhuma referência é feita à memória para buscar o operando. A desvantagem é o espaço de endereçamento muito limitado, afinal não existem tantos registradores, se comparado à memória principal.

Endereçamento indireto por registradores: análogo ao endereçamento indireto, sendo que a diferença é que no lugar de referência à memória, existe referência a um registrador:

$$EA = (R)$$

As vantagens e desvantagens são basicamente as mesmas do endereçamento indireto. O endereçamento indireto por registradores utiliza uma referência à memória a menos do que o endereçamento indireto.

Endereçamento por deslocamento: combina as capacidades do endereçamento direto e do endereçamento indireto por registradores:

$$EA = A + (R)$$

Esse modo de endereçamento requer que a instrução tenha dois campos de endereço, dos quais ao menos um seja explícito. O valor contido em um campo de endereço (valor = A) é utilizado diretamente e o outro campo de endereço refere-se a um registrador cujos conteúdos são adicionados a A para produzir um endereço efetivo. Vamos ver três dos usos mais comuns a seguir.

- **Endereçamento relativo**: o endereçamento é relativo ao registrador PC (*Program Counter*), ou seja, o endereço da próxima instrução é adicionado ao campo de endereço para produzir EA. Em geral, o campo de endereço é tratado como como um número complementar para essa operação. Assim, o endereço efetivo é o deslocamento relativo ao endereço da instrução;

- **Endereçamento por registrador base**: o registrador base contém um endereço da memória principal e o campo de endereço contém um deslocamento desse endereço (geralmente um inteiro sem sinal);

- **Indexação**: o campo de endereço faz referência a um endereço da memória principal e o registrador referenciado contém um deslocamento positivo desse endereço.

Endereçamento de pilha: itens são adicionados e retirados do topo da pilha, sendo que há um ponteiro cujo valor é o endereço do topo. O ponteiro da pilha é mantido em um registrador. O modo de endereçamento de pilha é uma forma de endereçamento implícito, sendo que as instruções de máquina não necessitam incluir uma referência de memória, devem apenas operar no topo da pilha.

Linguagem de Montagem

Como já vimos, um processador (CPU) entende executa instruções de máquina (opcodes e operandos, uma sequência de zeros e uns). Uma programação direto em linguagem de máquina seria muito complexo para um programador, a não ser que ele seja um robô, porque um ser humano prefere "palavras"!

Para melhorar um pouco a vida do programador (não muito, pois aí seria o caso de linguagens de alto nível), nomes simbólicos (mnemônicos) podem ser utilizados no lugar de cada instrução binária, mas ainda há o problema de utilizar endereços absolutos, afinal de contas quem sabe os endereços fixos que pode carregar os dados? E depois, para alterar alguma coisa, seria algo bem trabalhoso! Melhorando um pouco mais, há um sistema que utiliza endereços simbólicos, onde nós queremos chegar...a **linguagem de montagem (Assembly)**, aquela que pode ser traduzidas para linguagem de máquina através de um montador (*assembler*).

Em geral, cada instrução da linguagem de montagem é traduzida em uma instrução de máquina pelo montador (1:1). É uma linguagem **dependente do hardware**, ou seja, há uma linguagem de montagem diferente para cada tipo de processador. Um programador experiente em Assembly para a arquitetura RISC terá que aprender muita coisa para programar em Assembly para CISC, por exemplo (claro que a lógica ele não perde, mas as instruções são bem diferentes). Os quatro **elementos** da linguagem de montagem são mostrados a seguir.

Comentário (opcional): pode ser colocado no lado direito de um comando ou pode ocupar uma linha inteira. Em geral, o caractere especial que especifica que a partir dali rata-se de um comentário é o ponto e vírgula (;).

Rótulo (opcional): se estiver presente, o montador define o rótulo como equivalente ao endereço no qual o primeiro byte do código objeto gerado para essa instrução será carregado. O programador pode usar o rótulo como um endereço ou como dado no campo de endereço de outra instrução. Os rótulos são utilizados com mais frequência em instruções de desvio. Abaixo podemos ver um exemplo (rótulo foi denominado **L1** e os comentários são colocados após o ponto e vírgula).

```
L1:     SUB     EAX,    EDX     ; subtrai conteúdo do reg EDX do conteúdo de EAX e armazena o result em EAX

        JG      L1              ; salta para L1 se o resultado da subtração for positivo
```

Operando(s): uma sentença de linguagem de montagem inclui zero ou mais operandos. Cada operando identifica um valor imediato, um registrador ou uma posição de memória. Para endereçamento por registrador, o nome do registrador é usado, ex.: MOV ECX, EBX. O endereçamento imediato indica que o valor é codificado dentro da instrução, ex.: MOV EAX, 100H. O endereçamento direto refere-se a uma posição de memória e é expresso como um deslocamento a partir do registrador de segmento DS. Exemplo:

```
MOV AX, 1234H          ; AX <- 1234 (hexadecimal)

MOV [3420H], AX        ; conteúdo de AX é movido para o endereço lógico DS:3420H
```

Mnemônico: nome da operação ou função da sentença da linguagem de montagem. Uma sentença pode corresponder a uma instrução de máquina, uma diretiva do montador ou uma macro. No caso de uma instrução de máquina, um mnemônico é o nome simbólico associado com um determinado opcode. Vamos ver alguns mnemônicos, mas antes é importante conhecer os registradores de uso geral (arquitetura x86, a mais cobrada em concursos). Os oito registradores de uso geral são:

- EAX: acumulador, usado em operações aritméticas;
- ECX: contador, usado em *loops*;
- EDX: registrador de dados, usado em operações de entrada/saída e em multiplicações e divisões. É também uma extensão do acumulador;
- EBX: base, usado para apontar para dados no segmento DS (*data segment*);
- ESP: apontador da pilha (*Stack Pointer*), aponta para o topo da pilha (endereço mais baixo dos elementos da pilha);
- EBP: apontador da base do *frame*, usado para acessar argumentos de procedimentos passados pela pilha;
- ESI: índice da fonte de dados a copiar (*Source Index*), aponta para dados a copiar para DS:EDI;
- EDI: índice do destino de dados a copiar (*Destination Index*), aponta para o destino dos dados a copiar de DS:ESI.

Esses oito registradores possuem tamanho de 32 bits são "estendidos". Os 16 bits de ordem mais baixa de cada um dos registradores podem ser acessados através das versões não estendidas. As versões de 16 bits possuem os mesmos nomes que versões de 32 bits, com exceção de a letra E ser retirada (ex: EAX → AX). As versões estendidas dos registradores não existem em gerações anteriores à 80386 (primeira geração de processadores 32 bits da arquitetura x86).

As versões não estendidas dos quatro primeiros registradores de uso geral dividem-se ainda em dois grupos de 8 bits cada um. O octeto (byte) de ordem mais alta é acessado trocando o X por um H (exemplo: AX → AH), e o octeto de ordem mais baixa trocando o X por um L (ex.: AX → AL). Fica fácil lembrar quando sabemos que H = High (Alta) e L = Low (Baixa).

Agora vamos ver alguns dos mnemônicos mais conhecidos e seus significados:

- mov: move dados;
- add: adição aritmética;
- sub: subtração aritmética;
- push: empilha;
- pop: desempilha;

- jmp: salto incondicional;
- int: interrupção;
- call: chamada.

Alguns exemplos de como podem ser utilizados:

```
ADD AL, BL      ; AL <- AL + BL (AL e BL são os registradores com valores)

SUB AL, BL      ; AL <- AL - BL

MOV AL, 1       ; AL = 1 (move 1 para AL)
```

Por fim, vamos ver um exemplo de programa completo em Assembly, o clássico "Hello World!":

```
section .data
   msg db "Hello World!",0x0a     ; string "Hello world!"
   len equ $-msg                  ; calcula o tamanho da string msg
section .text                     ; início da seção de texto
   global _start                  ; onde deve começar a execução (assim como a main da ling. C)

_start:                           ; label start - a execução começa nesse ponto
   ; write
   mov ebx, 1                     ; arquivo de saída - stdin
   mov ecx, msg                   ; apontador para o buffer
   mov edx, len                   ; tamanho do buffer
   mov eax, 4                     ; chamada write ao sistema
   int 0x80                       ; chamada de sistema para o kernel

   ; exit
   mov eax, 1                     ; move o valor 1 para o registo eax
   mov ebx, 0                     ; move o valor 0 para o registo ebx
   int 0x80                       ; chamada de sistema para o kernel
```

Questões Comentadas

31. (VUNESP/Prefeitura de Ribeirão Preto-SP - 2018) Uma arquitetura de computador hipotética utiliza um microprocessador que possui instruções com o modo de endereçamento "endereçamento indireto por registrador". Considere a instrução de máquina a seguir, que utiliza esse tipo de endereçamento, envolvendo o registrador R1.

ADD A,(R1), 8

Considerando esse contexto, e que A representa o acumulador, 8 representa um valor imediato e ADD é o mnemônico de uma instrução de máquina que realiza a operação soma, assinale a alternativa que apresenta uma funcionalidade coerente para essa instrução e que utiliza o endereçamento indireto por registrador.

A) O resultado da soma do valor 8 com o valor do acumulador é armazenado no próprio acumulador.

B) O resultado da soma do valor que está em R1 com o valor do acumulador é armazenado no próprio acumulador.

C) O resultado da soma do valor 8 com o valor que está em R1 é armazenado no acumulador.

D) O resultado da soma do valor 8 com o dado que está na memória em um endereço apontado por R1 é armazenado no acumulador.

E) O resultado da soma do valor que está armazenado em R1 com o dado que está na memória de endereço 8 é armazenado no acumulador.

Comentários:

Quando há dois operandos, o resultado da soma (ADD) dos dois é colocado no primeiro operando. Mas a questão nos traz três, então a soma do segundo e do terceiro é colocada no primeiro (A).

Como R1 está entre parênteses (ou poderia ser colchetes também, dependendo do processador), indica que utiliza endereçamento indireto, ou seja, faz referência ao endereço de uma palavra na memória, a qual possui o endereço completo do operando.

O terceiro operando é o valor 8, simplesmente.

De tudo isso, podemos ler da seguinte forma a instrução ADD A,(R1), 8: o valor 8 é somado com o dado que está na memória em um endereço apontado por R1. O resultado é armazenado no acumulador (denominado A, pela questão).

Logo, a **alternativa D** está correta e é o gabarito da questão.

32. (FADESP/IF-PA - 2018) Em um sistema de computação, o modo mais simples de uma instrução especificar um operando é a parte da instrução referente ao endereço conter o operando de fato em vez de um endereço que descreva onde ele está. Ou seja, o operando é automaticamente buscado na memória, ao mesmo tempo que a própria instrução. Esse modo de endereçamento é denominado

A) imediato.

B) direto.

C) direto via registrador.

D) indireto.

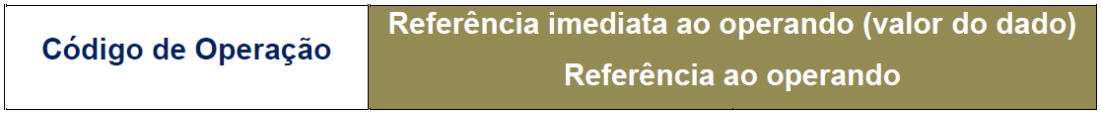
E) indexado.

Comentários:

Endereçamento imediato: é a forma mais simples de endereçamento, no qual o valor do operando está presente na instrução:

Operando = VALOR

Esse modo pode ser utilizado para definir e utilizar constantes ou definir valores iniciais das variáveis. A vantagem é que nenhuma referência de memória (além de obter a instrução em si) é necessária para obter o operando. Isso economiza um ciclo de memória ou de cache dentro do ciclo de instrução. A desvantagem é que o tamanho do número é limitado ao tamanho do campo de endereço (geralmente pequeno, se comparado ao tamanho da palavra, que é o tamanho utilizado quando se busca um dado da memória).



Um exemplo que utiliza o modo de endereçamento imediato para mover o valor 20_{16} para o registrador B:
MOV B, #20H

Logo, a **alternativa A** está correta e é o gabarito da questão.

33. (CESPE/ABIN - 2018) No método de endereçamento direto, a instrução contém o endereço da memória onde o dado está localizado.

Comentários:

Endereçamento direto: o campo de endereço possui o endereço efetivo do operando:

EA = A

Essa técnica era comum nas primeiras gerações de computadores, requer apenas uma referência à memória e nenhum cálculo especial. A limitação é que ela oferece um espaço de endereçamento limitado.

Logo, a questão está **correta**.

34. (FAURGS/BANRISUL - 2018) Assinale a alternativa que apresenta as características da instrução de movimentação "MVC PARM1,PARM2" na sua definição e execução.

A) Move o endereço do PARM2 para o endereço do PARM1.

B) Move o endereço do PARM1 para o endereço do PARM2.

C) Move o conteúdo do PARM1 para o local onde está PARM2.

D) Move o conteúdo de PARM2 para o local onde está PARM1.

E) Move o conteúdo do PARM2 para o endereço do PARM1.

Comentários:

Curiosidade: a instrução MVC permite a movimentação de 1 a 256 caracteres de uma localização na memória para outra. Mas esse não é o foco, pois poderia ter o MOV que a questão poderia ser respondida também! O foco é nos parâmetros. Em Assembly, temos que ler de trás para frente, ou seja, a movimentação ocorre do segundo operando para o primeiro, então de PARM2 para PARM1, conforme denominações dadas pela questão. Como não tem parênteses no PARM1 nem no PARM2, não estamos falando de endereços, e sim a movimentação do conteúdo de PARM2 para o local onde está PARM1 (mesma lógica quando lidamos com registradores). Logo, a **alternativa D** está correta e é o gabarito da questão.

35. (UFRR/UFRR - 2019) Quanto mais um programador dominar uma linguagem de programação, melhor ele conseguirá se expressar no mundo da programação e mais recursos ele terá para escrever soluções para problemas computacionais via código.

(trecho retirado de: www.universidadedatecnologia.com.br, acesso em 18/06/2019)

Supondo que o texto acima tem caráter unicamente motivador, responda:

Qual das alternativas abaixo NÃO representa uma linguagem de programação de alto nível:

A) C

B) C++

C) Assembly

D) JAVA

E) Visual Basic

Comentários:

Questão light, né? C, C++, Java e Visual Basic possuem comandos bem mais próximos do ser humano, são comandos em inglês em que um programador tem ideia do que ele faz e é mais tranquilo programar. São linguagens de alto nível. A linguagem de montagem (Assembly) possui mnemônicos para os opcodes da linguagem de máquina, lida diretamente com os registradores do processador, então podemos ver que não tem nada de alto nível! Logo, a **alternativa C** está correta e é o gabarito da questão.

36. (VUNESP/Câmara de Sertãozinho-SP - 2019) Em uma instrução de máquina, presente em uma arquitetura de computador, o modo direto de endereçamento é aquele em que no

A) campo operando da instrução está indicado o dado.

B) campo operando da instrução está indicado o endereço de memória, onde se localiza o dado.

C) campo operando da instrução está indicado o endereço de memória, onde se localiza o endereço do dado.

D) código de operação da instrução está indicado o dado.

E) código de operação da instrução está indicado o endereço de memória, onde se localiza endereço do dado.

Comentários:

Endereçamento direto: o campo de endereço possui o endereço efetivo do operando: EA = A.

Essa técnica era comum nas primeiras gerações de computadores, requer apenas uma referência à memória e nenhum cálculo especial. A limitação é que ela oferece um espaço de endereçamento limitado.

Logo, a **alternativa B** está correta e é o gabarito da questão.

Compilador, Montador, Interpretador e Ligador

Conceitos

Relembrando...as linguagens de programação de alto nível são aquelas mais próximas do ser humano. Alguns exemplos são ASP, C, Pascal, Visual Basic etc. A proporção entre as instruções de alto nível e as de baixo nível (linguagem de montagem) é 1:n, ou seja, um simples **printf("ola");** da linguagem C gera mais de uma instrução Assembly.

As linguagens de programação de baixo nível são próximas ao hardware, sendo que a linguagem de montagem (Assembly) possui uma relação 1:1 com o código objeto (binário). Se for necessária a união de mais de um código objeto (programa objeto), deve-se utilizar o ligador (*linkeditor*), sendo que a carga ocorre em tempo de compilação. Quando utilizadas as DLLs (*Dynamic Link Library*), a carga ocorre em tempo de execução. Ou seja, se for a união de algo "fixo" (códigos objeto), há uma ligação deles, e no caso de DLLs, os programas fazem a chamada delas somente quando são executados, deixando os executáveis menores.

Uma questão interessante é em relação à linguagem Java, que é compilada e interpretada. A compilação gera o Bytecode, que é interpretado pelas máquinas virtuais Java (JVMs).

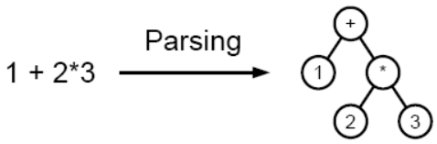
Compilador

De uma forma mais abrangente, um compilador serve para traduzir o código fonte de uma linguagem de programação de alto nível (C, Pascal etc.) para uma linguagem de programação de baixo nível (Assembly ou código de máquina). O candidato tem que pescar da banca como ela quer cobrar, se quer o entendimento alto nível → Assembly ou alto nível → código objeto. Abaixo vamos ver as etapas da forma mais completa, mas pode ser que o examinador nem cobre todas. Importante saber que existem as fases de **análise** (análise do código fonte) e **de síntese**. Coloquei em vermelho e verde, de acordo com a figura apresentada abaixo. Dentro de cada fase existem algumas etapas, que veremos a seguir.

Analisador léxico: tem como principal função a fragmentação do programa fonte em trechos elementares completos e com identidade própria (*tokens*). São eliminados os delimitadores e comentários, há a identificação de palavras reservada etc.

Há uma varredura no programa fonte da esquerda para a direita, agrupando os símbolos de cada item léxico e determinando a sua classe. Os *tokens* são representados por três propriedades: classe, valor e posição. Classe é uma palavra reservada, um operador aritmético, delimitadores, identificadores etc. Alguns exemplos de *token*, entre aspas: "if", "else", "(", ")", "+", "-".

Analisador sintático: tem como principal função promover a análise da sequência com que os átomos componentes do texto fonte se apresentam. Cria uma árvore sintática (árvore de derivação). A análise sintática (*parsing*) cuida exclusivamente da forma das sentenças da linguagem. Abaixo vemos um exemplo de uma árvore de derivação de uma expressão matemática que envolve adição e multiplicação.



Analisador semântico: tem como principal objetivo captar o significado das ações a serem tomadas no código fonte. Sua principal função é criar uma interpretação do texto, gerando uma linguagem intermediária. Algumas ações típicas são:

- manter informações sobre o escopo dos identificadores (global e local);
- validar tipos de dados, fluxos de controle e unicidade na declaração de variáveis (sabemos que não pode haver duas variáveis com o mesmo nome na mesma função).

Gerador de código intermediário: representação intermediária de um programa. Utiliza linguagem de baixo nível, porém fácil de produzir e de entender. Serve de ponte entre as duas fases (**análise** e **síntese**).

Otimizador de código: é uma etapa opcional que dificulta a engenharia reversa. A ideia principal é a otimização de:

- Tempo;
- Espaço;
- Consumo energético.

Gerador de código: converte as instruções de código intermediário em instruções da arquitetura. Gera o programa alvo. Ex.: geração de código para RISC ou CISC.

Alguns termos interessantes que já foram cobrados em provas de concurso são mostrados a seguir.

Compilador cruzado (*cross compiler*): produz código executável para uma plataforma diferente da qual o compilador está sendo executado. Ex.: compilador no Linux que gera código para o Windows.

Compilação JIT (*Just In Time*): tradução dinâmica (o próprio termo já deixa claro: "na hora"). Transforma um sistema híbrido em um sistema de compilação adiada (compilação de um programa em tempo de execução). Segundo a IBM, “o JIT é um componente do ambiente de tempo de execução que melhora o desempenho de aplicativos Java compilando bytecodes para o código de máquina nativo em tempo de execução”.

Montador e Ligador

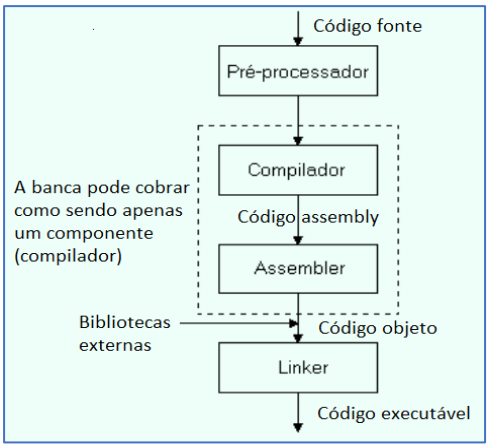
O **montador (*assembler*)** é o responsável pela tradução de código em linguagem Assembly para código objeto (binário). Em sistemas Unix-like, o código objeto geralmente é armazenado em arquivos com a extensão “.o”.

Dependendo da implementação, o conceito de montador pode estar integrado ao compilador, ou seja, o compilador "faz tudo", tendo como entrada um código fonte e entregando como resultado final o código objeto. Na prática isso é o mais comum! Mas para concursos, temos que saber bem o papel do montador!

O **ligador (*linker* ou *link-editor*)** combina todos os arquivos objeto **em tempo de compilação**, gerando o programa executável em linguagem de máquina. Por exemplo, o **#include** (linguagem de programação C) faz com que bibliotecas sejam incluídas no programa para que comandos que estão descritos nelas possam ser utilizados. Então, se o programador incluir alguma biblioteca desnecessária, o executável terá um tamanho maior e comandos dessa biblioteca nunca serão utilizados!

O *linker* não combina as DLLs (bibliotecas dinâmicas - códigos objeto utilizados por vários programas), pois estas são carregadas em tempo de execução.

Na figura ao lado podemos ver, de uma forma resumida, o "caminho" percorrido desde o código fonte até o executável. O pré-processador é um programa que faz alguns processamentos simples antes do compilador. Ele é executado automaticamente todas as vezes que o programa é compilado, e os comandos a serem executados são dados através de diretivas do pré-processador. Na linguagem C, as linhas que começam com um # são comandos para o pré-processador. Notamos ainda que o compilador gera o código assembly, que por sua vez passa pelo *assembler* (montador) para gerar o código objeto.



Como já vimos anteriormente, um compilador pode assumir o papel do *assembler* também, sendo um "2 em 1", mas isso você tem que "pescar" da questão. Tudo depende do examinador que a elaborou!

Vemos também na figura que, o código objeto, junto com bibliotecas (menos as bibliotecas dinâmicas - DLLs), são ligados através do *linker*, para gerar o código executável. Na prática, um compilador também já faz esse papel hoje em dia, mas lembre-se mais uma vez...estamos focando em concurso, e pode haver a cobrança de todos os elementos que vimos de forma separada!

Interpretador

Até há pouco estávamos falando de tradução em que tínhamos uma entrada e era gerada uma saída, ex.: código C compilado com o compilador gcc (que também "faz a parte do montador" e do ligador), gerando um programa executável.

Agora vamos ver o que é o interpretador...trata-se de um programa que "traduz instantaneamente" o código de programação de alto nível em código de máquina **sem criar um arquivo executável do programa traduzido**. Ou seja, a tradução acontece **instrução a instrução**, o que obviamente é mais lento, se comparado com a compilação. Alguns exemplos de linguagens interpretadas são BASIC, Perl e Python. Um exemplo de interpretador de shell (no Linux) é o Bash.

Questões Comentadas (Bancas variadas)

37. (FGV/SUSAM - 2014) Programa destinado a transformar um código escrito em linguagem de alto nível em uma linguagem Assembly é o

A) debugger.

B) compilador.

C) montador.

D) fortran.

E) otimizador.

Comentários:

Podemos ver que a banca optou pelo conceito mais abrangente, aquele que define que o compilador traduz do código fonte de alto nível para o Assembly (baixo nível). Depois deveria ser utilizado o montador para transformar o Assembly em código objeto e, se fosse necessário fazer alguma ligação com bibliotecas ou outros códigos objeto, deveria ser utilizado um ligador (*linker*). Portanto, a **alternativa B está correta e é o gabarito da questão**.

38. (FCC/TCE-GO - 2014) Compiladores, montadores e ligadores são softwares que convertem programas de um formato de código (entrada) para um mais próximo ao formato executável compreendido pela máquina (saída). Os ligadores geram como saída

A) programas objeto.

B) bibliotecas de programas semicompilados.

C) programas em formato bytecode.

D) programas executáveis em linguagem de máquina.

E) programas compilados em código intermediário, mas ainda não executáveis.

Comentários:

O ligador (*linker*) está lá no fim, depois do código fonte ter sido compilado e montado. Ele gera o código binário pronto para ser executado. Portanto, a **alternativa D está correta e é o gabarito da questão**.

39. (CESGRANRIO/CEFET-RJ - 2014) Um programador escolheu uma linguagem de alto nível para desenvolver uma aplicação para um cliente. Ele deseja entregar um código executável que possa ser simplesmente copiado na área de trabalho do cliente, que poderá executá-lo quando desejar, sem a necessidade de qualquer outro programa, recurso ou instalação, a não ser o sistema operacional (SO) nativo de sua máquina. Nessas circunstâncias, o programador necessitará de um

A) tradutor capaz de gerar código para uma máquina virtual que executará o programa.

B) montador (assembler) capaz de gerar código de máquina para a plataforma e SO do cliente, a partir de um código de montagem (assembly).

C) editor integrado em um ambiente de desenvolvimento para a plataforma do programador, instalado em uma máquina virtual apenas no ambiente do cliente.

D) ligador (linkeditor) capaz de unir o código objeto da plataforma do programador com as bibliotecas existentes apenas na plataforma e SO do cliente.

E) compilador capaz de gerar código executável para a plataforma e SO do cliente.

Comentários:

Nessa questão o examinador utilizou o conceito mais simplificado de compilador, pois ele "pega" o código fonte e já entrega o executável pronto para ser executado no sistema operacional o qual o compilador funciona (ex.: compilador no Windows gerando um software para o Windows). Notamos que esse compilador é aquele que tem "embutidas" as funcionalidades do montador e do ligador também. Portanto, a **alternativa E está correta e é o gabarito da questão**.

40. (CETRO/AMAZUL - 2015) Na compilação de um programa, assinale a alternativa que apresenta a etapa/fase em que ocorre a geração de um programa executável.

A) Montagem.

B) Compilação.

C) Linkedição.

D) Interpretação.

E) Carregador.

Comentários:

Percebemos que essa questão é detalhista, ou seja, o compilador não gera direto o executável. Então vamos ver a sequência: código fonte → compilador → montador → ligador (link-editor) → executável.

Portanto, a **alternativa C está correta e é o gabarito da questão**.

41. (FCC/TRT-14ª Região - 2016) A compilação é o processo de tradução de um programa escrito em uma linguagem fonte em um programa equivalente em linguagem de máquina. Nesse processo, o programa fonte normalmente passa pelas fases:

I. Identificação de sequências de caracteres de entrada e produção de uma sequência de elementos de saída, os tokens. Nesta fase, verifica-se se cada caractere do programa fonte pertence ao alfabeto da linguagem, identificando os tokens e desprezando comentários e espaços em branco. Os tokens constituem classes de símbolos, tais como palavras reservadas, delimitadores, identificadores etc.

II. Identificação de sequências de símbolos que constituem estruturas como expressões e comandos, através de uma varredura, ou parsing, da representação interna do programa fonte, produzindo uma estrutura em árvore, chamada árvore de derivação.

III. Verificação das estruturas quanto ao sentido, ou seja, se o programa não possui erros de significado. Por exemplo, verifica se um identificador declarado como variável é utilizado como tal, se existe compatibilidade entre operandos e operadores em expressões etc.

Os itens I, II e III referem-se, correta e respectivamente, às fases

A) Análise Léxica − Análise Sintática − Análise Semântica.

B) Interpretação − Análise Sintática − Montagem.

C) Busca Binária − Montagem Léxica − Análise Semântica.

D) Classificação − Análise Léxica − Montagem.

E) Identificação Inicial − Análise Estrutural − Geração de Código.

Comentários:

I. Quando o foco é em tokens, estamos falando da análise léxica. II. Quando o foco é em varredura (*parsing*) para a produção de uma estrutura em árvore (árvore de derivação), estamos falando da análise sintática. III. Quando há uma busca pelo sentido/significado, trata-se da análise semântica. Portanto, a **alternativa A está correta e é o gabarito da questão**.

42. (COPEVE-UFAL/UFAL - 2016) Considere as afirmativas:

I. cria o código objeto traduzindo as instruções da linguagem de montagem (assembly) para código de máquina;

II. recebe como entrada um conjunto de arquivos objetos e bibliotecas, e produz como resultado um arquivo objeto de saída;

III. traduz um programa descrito em uma linguagem de alto nível para um programa em linguagem simbólica ou linguagem de máquina;

IV. recebe uma instrução do programa fonte, converte-a em linguagem de máquina e ordena ao computador que execute esta instrução.

Nessa ordem, os itens de I a IV referem-se a

A) ligador, montador, interpretador e montador.

B) ligador, montador, compilador e interpretador.

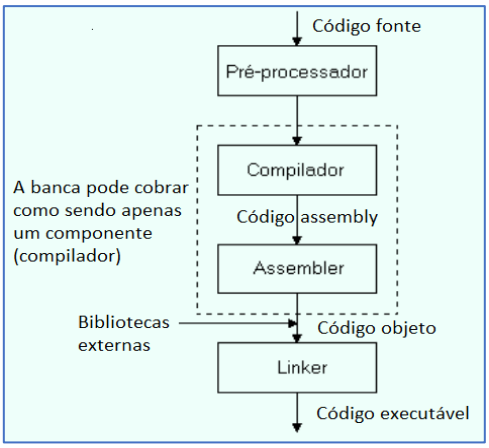
C) interpretador, ligador, compilador e montador.

D) montador, ligador, compilador e interpretador.

E) compilador, ligador, montador e interpretador.

Comentários:

Ao lado podemos ver uma figura com a compilação, a montagem e a ligação.



Em relação ao interpretador, é aquele programa que não gera um executável, ele simplesmente vai traduzindo e executando instrução por instrução na medida em que vai lendo o código fonte.

Portanto, a **alternativa D está correta e é o gabarito da questão**.

43. (FCC/TRF-3ª Região - 2016) Um compilador

A) baseado em um sistema de compilação pura não realiza traduções; os programas são interpretados por um software interpretador.

B) do tipo JIT- Just in Time tem sido utilizado em linguagens como Java e C#. Na prática, JIT transforma um sistema híbrido em um sistema de compilação adiada.

C) que utiliza implementação híbrida traduz o programa para um código de formato intermediário e este código é compilado.

D) separa a tarefa de analisar a semântica do código em 2 partes: análise léxica e análise sintática. Este processo é realizado por uma máquina virtual.

E) de uma linguagem de programação traduz código de baixo nível para linguagem de máquina.

Comentários:

Compilação JIT (*Just In Time*): tradução dinâmica (o próprio termo já deixa claro: "na hora"). Transforma um sistema híbrido em um sistema de compilação adiada (compilação de um programa em tempo de execução). Segundo a IBM, “o JIT é um componente do ambiente de tempo de execução que melhora o desempenho de aplicativos Java compilando bytecodes para o código de máquina nativo em tempo de execução”.

Portanto, a **alternativa B está correta e é o gabarito da questão**.

44. (FGV/IBGE - 2017) O módulo de análise léxica de um compilador tem por objetivo:

A) verificar se o programa-fonte obedece às regras da gramática da linguagem;

B) agrupar coerentemente os caracteres do programa-fonte em tokens;

C) gerar o código objeto correspondente à tradução do programa-fonte para alguma forma intermediária de representação;

D) construir as árvores sintáticas dos diversos comandos do programa-fonte;

E) eliminar comandos supérfluos do programa-fonte.

Comentários:

C) gerador de código; D) análise sintática; E) otimização.

Portanto, a **alternativa B está correta e é o gabarito da questão**.

45. (FGV/MPE-AL - 2018) Na implementação de compiladores, a fase de parser do programa baseia-se, em parte, no resultado de um analisador léxico. Assinale a opção que descreve o papel de um analisador léxico.

A) Representar as regras da gramática da linguagem.

B) Verificar a conformidade do código fonte com as regras da gramática da linguagem.

C) Definir a notação em que as regras da gramática são expressas.

D) Identificar os tokens gramaticais no código fonte.

E) Exprimir a semântica das construções da linguagem.

Comentários:

Palavra-chave = tokens → Analisador léxico! Portanto, a **alternativa D está correta e é o gabarito da questão**.

46. (SUGEP-UFRPE/UFRPE - 2018) Em relação a compiladores, ligadores (link-editores) e interpretadores, é correto afirmar que:

A) uma função importante do tradutor é a realocação.

B) o link-editor é o utilitário responsável por carregar, na memória principal, um programa para ser executado.

C) o depurador é o utilitário responsável por gerar, a partir de um ou mais módulos-objeto, um único programa executável.

D) o link-editor é o utilitário responsável por gerar, a partir de um ou mais módulos-objeto, um único programa executável.

E) a grande vantagem do uso de interpretadores é o tempo gasto na tradução de suas instruções sempre que é executado.

Comentários:

O ligador (link-editor) é aquele "lá no final", que une códigos objeto em um único arquivo executável. Daí o nome ligador! Portanto, a **alternativa D está correta e é o gabarito da questão**.

47. (SUGEP-UFRPE/UFRPE - 2018) Abaixo, estão enumeradas as fases que integram o front-end de um compilador:

1) Análise Semântica

2) Análise Léxica

3) Análise Sintática

4) Gerador de código intermediário

Indique a sequência correta, com a ordem em que as fases ocorrem.

A) 1, 3, 2 e 4.

B) 3, 1, 4 e 2.

C) 2, 3, 1 e 4.

D) 1, 4, 2 e 3.

E) 4, 1, 2 e 3.

Comentários:

2, 3, 1 e 4

Verifica os tokens (análise léxica) → Realiza a varredura (*parsing*) para montar a árvore de derivação (análise sintática) → verifica o significado (tipo de dados, fluxo) (análise semântica) → gera o código intermediário. Portanto, a **alternativa C está correta e é o gabarito da questão**.

48. (IBADE/Câmara de Porto Velho-RO - 2018) Algumas linguagens exigem que o código fonte seja previamente traduzido para linguagem de máquina antes de ser executado. Chama-se esta fase de:

A) Linkedição.

B) Interpretação.

C) Tradução.

D) Compilação.

E) Edição.

Comentários:

O examinador usou o conceito mais simples, aquele em que o compilador possui "embutidos" o assembler e o link-editor, tendo como entrada o código fonte e como saída o arquivo executável. Portanto, a **alternativa D está correta e é o gabarito da questão**.

49. (COMPERVE/UFRN - 2019) O conjunto de instruções de uma arquitetura de computador, ou Instruction Set Architecture - ISA, define as operações que podem ser utilizadas em um programa para ser executado nessa arquitetura. Um ISA define também

A) a linguagem de programação específica na qual o programa deve ser escrito.

B) o compilador específico que deve ser utilizado para compilar o programa.

C) o formato dos bits das instruções que são apresentadas ao processador.

D) a quantidade de estágios de pipeline que a arquitetura deve ter para executar o programa.

Comentários:

O conjunto de instruções de uma arquitetura (ISA) define as instruções e os seus formatos. Não tem nada a ver com compilador! Só coloquei aqui como pegadinha mesmo! Portanto, a **alternativa C está correta e é o gabarito da questão**.

Questões Comentadas (Banca CESPE)

50. (CESPE/ANATEL - 2014) A compilação é o processo de análise de um programa escrito em linguagem de alto nível, denominado programa-fonte, e sua conversão em um programa equivalente, escrito em linguagem binária de máquina, denominado programa-objeto.

Comentários:

Tendo em consideração aquele conceito mais simples, está ok. Por exemplo, se você pegar o gcc, que é um compilador bastante conhecido, é só passar como entrada um programa feito em C que ele entrega o executável pronto para ser utilizado. Portanto, a questão está **correta**.

51. (CESPE/TCE-PA - 2016) Na compilação de um programa que chama uma biblioteca compartilhada, todo o código da biblioteca é copiado e inserido dentro do binário final.

Comentários:

Uma biblioteca compartilhada (ex.: DLL, no Windows) não é compilada junto com o código fonte. O executável faz chamadas à DLL, inclusive se a DLL for atualizada, as chamadas serão feitas para a nova DLL sem haver a necessidade de nova compilação! Portanto, a questão está **errada**.

52. (CESPE/TCE-PA - 2016) Os interpretadores, em vez de produzirem um programa objeto, fruto da tradução, executam diretamente as operações especificadas no código-fonte.

Comentários:

Enquanto um compilador traduz o código fonte em um código binário (o montador também faz isso), o interpretador traduz instrução por instrução "online", sem gerar nenhum código executável. Portanto, a questão está **correta**.

53. (CESPE/TCE-PA - 2016) Compilador é um utilitário responsável por executar um programa objeto diretamente na máquina.

Comentários:

O compilador pode ser visto como responsável por:

- traduzir um código fonte de alto nível em um código de montagem (Assembly) e um montador traduz o Assembly em código objeto;
- traduzir o código fonte direto para o executável "final" (como se tivesse embutido um assembler e um linker).

Portanto, a questão está **errada**.

54. (CESPE/ABIN - 2018) Chama-se cruzado o compilador que gera um programa que seja executável em pelo menos um sistema operacional diferente daquele onde o compilador tenha sido executado.

Comentários:

Compilador cruzado (*cross compiler*): produz código executável para uma plataforma diferente da qual o compilador está sendo executado. Ex.: compilador no Linux que gera código para o Windows.

Portanto, a questão está **correta**.

55. (CESPE/ABIN - 2018) Em um compilador, os tokens são identificados na fase de análise léxica e são representados por três propriedades: classe, valor e posição.

Comentários:

Analisador léxico: tem como principal função a fragmentação do programa fonte em trechos elementares completos e com identidade própria (*tokens*). São eliminados os delimitadores e comentários, há a identificação de palavras reservada etc.

Há uma varredura no programa fonte da esquerda para a direita, agrupando os símbolos de cada item léxico e determinando a sua classe. Os *tokens* são representados por três propriedades: classe, valor e posição. Classe é uma palavra reservada, um operador aritmético, delimitadores, identificadores etc. Alguns exemplos de *token*, entre aspas: "if", "else", "(", ")", "+", "-".

Portanto, a questão está **correta**.

56. (CESPE/ABIN - 2018) A etapa de análise semântica de um compilador tem como objetivo verificar os inter-relacionamentos de um programa, validando tipologias, fluxos de controle e unicidade na declaração de variáveis.

Comentários:

Analisador semântico: tem como principal objetivo captar o significado das ações a serem tomadas no código fonte. Sua principal função é criar uma interpretação do texto, gerando uma linguagem intermediária. Algumas ações típicas são:

- manter informações sobre o escopo dos identificadores (global e local);
- validar tipos de dados, fluxos de controle e unicidade na declaração de variáveis (sabemos que não pode haver duas variáveis com o mesmo nome na mesma função).

Portanto, a questão está **correta**.

57. (CESPE/EBSERH - 2018) Compilador é o programa que traduz o código fonte de uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível.

Comentários:

Agora apareceu uma questão que cobra o conceito mais "detalhista", ou seja, que o compilador traduz de uma linguagem de alto nível (ex.: C) em uma linguagem de baixo nível (linguagem de montagem - Assembly). A partir daqui quem assume é o assembler e o linker. Portanto, a questão está **correta**.

LISTA DE QUESTÕES

1. (FUNIVERSA/IPHAN - 2009) Um sistema de processamento de dados é composto, basicamente, por três etapas: (1) entrada de dados, (2) processamento ou tratamento da informação e (3) saída. Em um computador, essas tarefas são realizadas por partes diversas que o compõem, como teclado, mouse, microprocessador, memória etc. Levando-se em conta as tarefas de processamento de dados realizadas por um computador, é correto afirmar que

A) dispositivos de hardware como teclado e mouse são responsáveis pela saída de dados, uma vez que escrevem ou apontam o resultado esperado em uma operação realizada pelo computador.

B) acessórios modernos como webcams, bluetooth e leitores biométricos são dispositivos de saída de dados incorporados a alguns computadores como acessórios de fábrica.

C) a tela (ou monitor) de um computador comporta-se como um dispositivo de entrada de dados, quando se trabalha em sistemas de janelas, com botões a serem “clicados” pelo usuário.

D) as impressoras multifuncionais são dispositivos mistos, de entrada, processamento e saída de dados, pois podem ler (scanner), processar (memória interna) e imprimir informações.

E) a entrada de dados é tarefa realizada pela pessoa (ou por um programa de computador) responsável por alimentar o sistema com dados necessários para atingir o resultado esperado.

2. (MS CONCURSOS/CODENI-RJ - 2010) É o componente vital do sistema, porque, além de efetivamente realizar as ações finais, interpreta o tipo e o modo de execução de uma instrução, bem como controla quando e o que deve ser realizado pelos demais componentes, emitindo para isso sinais apropriados de controle. A descrição acima refere-se a?

A) Dispositivos de Entrada e Saída.

B) Memória Principal.

C) Memória Secundária.

D) Unidade Central de Processamento.

3. (CESPE/EBC - 2011) São funções básicas de um computador: processamento de dados, armazenamento de dados, transferência de dados e controle. São componentes estruturais de um computador: unidade central de processamento, memória principal, dispositivos de entrada e saída e sistemas de interconexão.

4. (AOCP/TCE-PA - 2012) Em computação CPU significa

A) Central de Processamento Única.

B) Único Centro de Processamento.

C) Unidade Central de Processamento.

D) Central da Unidade de Processamento.

E) Centro da Unidade de Processamento.

5. (MS CONCURSOS/IF-AC - 2014) Dentre as funções básicas do computador, podemos citar, exceto:

A) Entrada de dados.

B) Processamento de Dados.

C) Saída de Informações.

D) Capacidade de Unidade.

6. (CESPE/Polícia Científica-PE - 2016) Assinale a opção correta acerca da arquitetura Harvard de microprocessadores.

A) É a arquitetura mais antiga em termos de uso em larga escala

B) Não permite pipelining.

C) Não permite o uso de um conjunto reduzido de instruções.

D) Dispensa a unidade lógica aritmética

E) Apresenta memórias de programa e de dados distintas e independentes em termos de barramentos.

7. (UFMT/UFSBA - 2017) A respeito de memória cache, os projetos denominados arquitetura Harvard são aqueles

A) cuja cache é unificada, com dados e instruções na mesma cache.

B) cujos conceitos do princípio da localidade foram descartados e adotou-se um protocolo serial de acesso a dados.

C) cuja cache é dividida, com instruções em uma e os dados em outra.

D) cujo empacotamento de módulos de memória cache foi colocado fora do chip, reduzindo o custo de produção e aumentando a quantidade de memória disponível.

8. (UFPA/UFPA - 2017) O gargalo de von Neumann é caracterizado pela maior velocidade de processamento do processador em relação ao que a memória pode servir a ele. Para minimizar esse gargalo, é necessário

A) utilizar sempre as versões mais atualizadas dos sistemas operacionais.

B) utilizar memória cache entre o processador e a memória principal com caminhos separados para dados e instruções.

C) utilizar processadores de 32 bits ao invés de 64 bits.

D) aplicar o processo de desfragmentação do disco.

E) bloquear a utilização de algoritmos e lógicas de branchpredictor.

9. (INAZ do Pará/CFF - 2017) A arquitetura de computadores de Von Neumann é frequentemente definida como o conjunto de atributos da máquina que um programador deve compreender para que consiga programar o computador específico com sucesso, e também são compostas de três subsistemas básicos. Assinale a alternativa correta que apresenta os três subsistemas básicos.

A) CPU, memória principal e sistema de entrada e saída.

B) Vídeo, memória externa e não volátil e sistema de entrada e saída.

C) CPU, memória secundária e sistema de entrada e saída.

D) CPU, memória principal e sistema operacional.

E) Vídeo, memória secundária e sistema de entrada e saída.

10. (CESPE/ABIN - 2018) Na arquitetura de Von Neumann, o caminho único de dados é o barramento físico, que liga a memória diretamente aos dispositivos de entrada e saída (E/S): o objetivo desse barramento é a troca de dados externos com a máquina, enquanto a memória guarda os dados de forma temporária no computador.

11. (COPESE-UFT/UFT - 2018) Em 1952 John von Neumann desenvolveu um protótipo de um novo computador de programa armazenado. Esse projeto ficou conhecido como arquitetura de Von Neumann e ainda hoje influencia o projeto de computadores modernos. Os componentes abaixo fazem parte da arquitetura de Von Neumann, EXCETO:

A) Memória Principal.

B) Unidade Lógica e Aritmética (ALU).

C) Barramento.

D) Equipamento de Entrada e Saída (E/S).

12. (INSTITUTO PRÓ-MUNICÍPIO/CRP-11ª Região - 2019) O computador é uma máquina que processa informações eletronicamente, na forma de dados e pode ser programado para as mais diversas tarefas. As fases do processamento são:

A) Monotarefa, Monousuário e Multitarefa;

B) Entrada de dados, Processamento e Saída de Dados;

C) Operação, Linguagem e Aplicação;

D) Programação, Instalação e Registro de Dados.

13. (ESAF/SUSEP - 2010) Em uma Arquitetura RISC

A) há poucos registradores.

B) há pouco uso da técnica pipelining.

C) as instruções possuem diversos formatos.

D) as instruções são realizadas por microcódigo.

E) as instruções utilizam poucos ciclos de máquina.

14. (FCC/TRE-AM - 2010) Numa máquina estruturada multinível, é o nível essencial para as máquinas CISC (Complex Instruction Set Computer), mas que inexiste nas máquinas RISC (Reduced Instruction Set Computer). Trata-se do nível

A) do sistema operacional.

B) de lógica digital.

C) de microprogramação.

D) convencional de máquina.

E) do montador.

15. (CESPE/Correios - 2011) As instruções CISC são mais simples que as instruções RISC, por isso, os compiladores para máquinas CISC são mais complexos, visto que precisam compensar a simplificação presente nas instruções. Entretanto, se for usado pipeline, a complexidade do compilador CISC é reduzida, pois a arquitetura pipeline evita a necessidade de reordenação inteligente de instruções.

16. (VUNESP/UNESP - 2013) Um computador baseado em uma Unidade Central de Processamento do tipo RISC

A) não faz uso de pipeline.

B) executa cada instrução em um ciclo de relógio

C) possui instruções de tamanho variável.

D) possui muitos modos de endereçamento

E) possui um grande conjunto de instruções.

17. (FUNDEP/IPSEMG - 2013) A arquitetura RISC de um computador possui as seguintes características, EXCETO:

A) Formatos simples de instruções.

B) Modos simples de endereçamento.

C) Operações memória-para-memória.

D) Uma instrução por ciclo.

18. (CESPE/Antaq - 2014) Atualmente, os fabricantes de computadores têm adotado exclusivamente a arquitetura RISC para o desenvolvimento de chips para processadores, dado o melhor desempenho dessa arquitetura em relação à arquitetura CISC.

19. (IADES/PCDF - 2016) Em relação ao projeto de máquinas RISC e CISC, assinale a alternativa correta.

A) Dadas as características das instruções das máquinas CISC, o pipeline fica favorecido nessa arquitetura.

B) Arquiteturas RISC normalmente realizam poucas operações de registrador para registrador, aumentando o acesso à memória cache.

C) Programas para arquiteturas CISC sempre possuem tamanho menor que programas para arquiteturas RISC, devido à relação um para um de instruções de máquina e instruções de compilador.

D) Arquiteturas RISC tendem a enfatizar referências aos registradores no lugar de referências à memória.

E) Arquiteturas CISC usam um número muito grande de instruções simples em detrimento de instruções complexas.

20. (INAZ do Pará/CORE-SP - 2019) "O projeto do Conjunto de Instruções inicia com a escolha de uma entre duas abordagens, a abordagem RISC e a CISC".

Disponível em: http://producao.virtual.ufpb.br/books/edusantana/introducao-a-arquitetura-de-computadores-livro/livro/livro.chunked/ch04s04.html. Acesso em: 13.12.2018.

Quais são características do paradigma RISC de projeto de CPU?

A) São mais baratos, menos acesso à memória, conjunto de instruções simples.

B) Objetivo de criar um hardware mais otimizado, com isso os programas tendem a ocupar menos espaço em memória.

C) Grande número de registradores de propósito geral e os programas tendem a ocupar menos espaço em memória.

D) Em geral usa mais memória para armazenamento de dados.

E) Muitos modos de endereçamento, e foco no hardware.

21. (Quadrix/CRA-PR - 2019) Possuir um conjunto de instruções simples e limitado é uma das principais características da arquitetura CISC.

22. (Quadrix/CRA-PR - 2019) A característica que mais se destaca na arquitetura RISC é que computadores pertencentes a ela realizam milhares de instruções por ciclo.

23. (Quadrix/CREA-GO - 2019) Uma máquina RISC, geralmente, usa um conjunto de modos de endereçamento relativamente simples e direto.

24. (CESPE/TJ-PA - 2021) Na tentativa de solucionar o chamado espaço semântico (semantic gap), fabricantes de computadores de grande porte criaram alternativas para resolver o problema, como, por exemplo,

I maior densidade de código a ser executado.

II utilização em larga escala do pipelining.

III execução otimizada de chamadas de funções via registradores.

A arquitetura CISC contempla

A) apenas o item I.

B) apenas o item II.

C) apenas os itens I e III.

D) apenas os itens II e III.

E) todos os itens.

25. (CESPE/TCE-PA - 2016) Utilizando-se linguagens fracamente tipadas, é possível alterar o tipo de dado contido em uma variável durante a execução do programa.

26. (CESPE/TCE-PA - 2016) Acerca de funções e procedimentos em subprogramas, julgue o item que se segue.

```
algoritmo solução1
var
   A, B, X : inteiro
início
   leia (A, B)
     X ← A
     A ← B
     B ← X
   escreva (A, B)
Fim algoritmo.
algoritmo solução2
var
  A, B : inteiro
Procedimento TROCA
  var
     X : inteiro
  início
     X ← A
     A ← B
     B ← X
  fim
início
    leia (A, B)
     TROCA
     escreva (A, B)
Fim algoritmo.
```

No algoritmo solução1 apresentado a seguir as variáveis X, A e B são criadas com escopo global; no algoritmo solução2 apresentado após algoritmo solução1, as variáveis A e B são criadas com escopo global e a variável X com escopo local.

27. (FUNDEP/UFVJM-MG - 2017) Assinale a alternativa que apresenta corretamente a sequência de passos computacionais que transforma a entrada na saída, ou seja, procedimentos necessários para resolver um determinado problema.

A) Algoritmos

B) Arquivos

C) Cases

D) Polinômio

28. (CESPE/TRE-TO - 2017) Assinale a opção que apresenta o resultado final após a execução do algoritmo precedente.

```
algoritmo
var numero: inteiro
inicio
numero = 12

   se (numero mod 2 = 0) entao
      escreva ("A")
   senao
      escreva ("B")
   fim-se

   se (numero > 12) entao
      escreva ("C")
   fim-se

fim
```

A) B

B) A

C) AC

D) C

E) BC

29. (Quadrix/CRM-PR - 2018) Em um fluxograma, as caixas de decisão são como "caixas pretas", uma vez que não se tem clareza da ação que será executada.

30. (Quadrix/CRM-PR - 2018) Os algoritmos são sequências finitas de instruções que, quando corretamente executadas, levam à solução de um problema.

31. (VUNESP/Prefeitura de Ribeirão Preto-SP - 2018) Uma arquitetura de computador hipotética utiliza um microprocessador que possui instruções com o modo de endereçamento "endereçamento indireto por registrador". Considere a instrução de máquina a seguir, que utiliza esse tipo de endereçamento, envolvendo o registrador R1.

ADD A,(R1), 8

Considerando esse contexto, e que A representa o acumulador, 8 representa um valor imediato e ADD é o mnemônico de uma instrução de máquina que realiza a operação soma, assinale a alternativa que apresenta uma funcionalidade coerente para essa instrução e que utiliza o endereçamento indireto por registrador.

A) O resultado da soma do valor 8 com o valor do acumulador é armazenado no próprio acumulador.

B) O resultado da soma do valor que está em R1 com o valor do acumulador é armazenado no próprio acumulador.

C) O resultado da soma do valor 8 com o valor que está em R1 é armazenado no acumulador.

D) O resultado da soma do valor 8 com o dado que está na memória em um endereço apontado por R1 é armazenado no acumulador.

E) O resultado da soma do valor que está armazenado em R1 com o dado que está na memória de endereço 8 é armazenado no acumulador.

32. (FADESP/IF-PA - 2018) Em um sistema de computação, o modo mais simples de uma instrução especificar um operando é a parte da instrução referente ao endereço conter o operando de fato em vez de um endereço que descreva onde ele está. Ou seja, o operando é automaticamente buscado na memória, ao mesmo tempo que a própria instrução. Esse modo de endereçamento é denominado

A) imediato.

B) direto.

C) direto via registrador.

D) indireto.

E) indexado.

33. (CESPE/ABIN - 2018) No método de endereçamento direto, a instrução contém o endereço da memória onde o dado está localizado.

34. (FAURGS/BANRISUL - 2018) Assinale a alternativa que apresenta as características da instrução de movimentação "MVC PARM1,PARM2" na sua definição e execução.

A) Move o endereço do PARM2 para o endereço do PARM1.

B) Move o endereço do PARM1 para o endereço do PARM2.

C) Move o conteúdo do PARM1 para o local onde está PARM2.

D) Move o conteúdo de PARM2 para o local onde está PARM1.

E) Move o conteúdo do PARM2 para o endereço do PARM1.

35. (UFRR/UFRR - 2019) Quanto mais um programador dominar uma linguagem de programação, melhor ele conseguirá se expressar no mundo da programação e mais recursos ele terá para escrever soluções para problemas computacionais via código.

(trecho retirado de: www.universidadedatecnologia.com.br, acesso em 18/06/2019)

Supondo que o texto acima tem caráter unicamente motivador, responda:

Qual das alternativas abaixo NÃO representa uma linguagem de programação de alto nível:

A) C

B) C++

C) Assembly

D) JAVA

E) Visual Basic

36. (VUNESP/Câmara de Sertãozinho-SP - 2019) Em uma instrução de máquina, presente em uma arquitetura de computador, o modo direto de endereçamento é aquele em que no

A) campo operando da instrução está indicado o dado.

B) campo operando da instrução está indicado o endereço de memória, onde se localiza o dado.

C) campo operando da instrução está indicado o endereço de memória, onde se localiza o endereço do dado.

D) código de operação da instrução está indicado o dado.

E) código de operação da instrução está indicado o endereço de memória, onde se localiza endereço do dado.

37. (FGV/SUSAM - 2014) Programa destinado a transformar um código escrito em linguagem de alto nível em uma linguagem Assembly é o

A) debugger.

B) compilador.

C) montador.

D) fortran.

E) otimizador.

38. (FCC/TCE-GO - 2014) Compiladores, montadores e ligadores são softwares que convertem programas de um formato de código (entrada) para um mais próximo ao formato executável compreendido pela máquina (saída). Os ligadores geram como saída

A) programas objeto.

B) bibliotecas de programas semicompilados.

C) programas em formato bytecode.

D) programas executáveis em linguagem de máquina.

E) programas compilados em código intermediário, mas ainda não executáveis.

39. (CESGRANRIO/CEFET-RJ - 2014) Um programador escolheu uma linguagem de alto nível para desenvolver uma aplicação para um cliente. Ele deseja entregar um código executável que possa ser simplesmente copiado na área de trabalho do cliente, que poderá executá-lo quando desejar, sem a necessidade de qualquer outro programa, recurso ou instalação, a não ser o sistema operacional (SO) nativo de sua máquina. Nessas circunstâncias, o programador necessitará de um

A) tradutor capaz de gerar código para uma máquina virtual que executará o programa.

B) montador (assembler) capaz de gerar código de máquina para a plataforma e SO do cliente, a partir de um código de montagem (assembly).

C) editor integrado em um ambiente de desenvolvimento para a plataforma do programador, instalado em uma máquina virtual apenas no ambiente do cliente.

D) ligador (linkeditor) capaz de unir o código objeto da plataforma do programador com as bibliotecas existentes apenas na plataforma e SO do cliente.

E) compilador capaz de gerar código executável para a plataforma e SO do cliente.

40. (CETRO/AMAZUL - 2015) Na compilação de um programa, assinale a alternativa que apresenta a etapa/fase em que ocorre a geração de um programa executável.

A) Montagem.

B) Compilação.

C) Linkedição.

D) Interpretação.

E) Carregador.

41. (FCC/TRT-14ª Região - 2016) A compilação é o processo de tradução de um programa escrito em uma linguagem fonte em um programa equivalente em linguagem de máquina. Nesse processo, o programa fonte normalmente passa pelas fases:

I. Identificação de sequências de caracteres de entrada e produção de uma sequência de elementos de saída, os tokens. Nesta fase, verifica-se se cada caractere do programa fonte pertence ao alfabeto da linguagem, identificando os tokens e desprezando comentários e espaços em branco. Os tokens constituem classes de símbolos, tais como palavras reservadas, delimitadores, identificadores etc.

II. Identificação de sequências de símbolos que constituem estruturas como expressões e comandos, através de uma varredura, ou parsing, da representação interna do programa fonte, produzindo uma estrutura em árvore, chamada árvore de derivação.

III. Verificação das estruturas quanto ao sentido, ou seja, se o programa não possui erros de significado. Por exemplo, verifica se um identificador declarado como variável é utilizado como tal, se existe compatibilidade entre operandos e operadores em expressões etc.

Os itens I, II e III referem-se, correta e respectivamente, às fases

A) Análise Léxica − Análise Sintática − Análise Semântica.

B) Interpretação − Análise Sintática − Montagem.

C) Busca Binária − Montagem Léxica − Análise Semântica.

D) Classificação − Análise Léxica − Montagem.

E) Identificação Inicial − Análise Estrutural − Geração de Código.

42. (COPEVE-UFAL/UFAL - 2016) Considere as afirmativas:

I. cria o código objeto traduzindo as instruções da linguagem de montagem (assembly) para código de máquina;

II. recebe como entrada um conjunto de arquivos objetos e bibliotecas, e produz como resultado um arquivo objeto de saída;

III. traduz um programa descrito em uma linguagem de alto nível para um programa em linguagem simbólica ou linguagem de máquina;

IV. recebe uma instrução do programa fonte, converte-a em linguagem de máquina e ordena ao computador que execute esta instrução.

Nessa ordem, os itens de I a IV referem-se a

A) ligador, montador, interpretador e montador.

B) ligador, montador, compilador e interpretador.

C) interpretador, ligador, compilador e montador.

D) montador, ligador, compilador e interpretador.

E) compilador, ligador, montador e interpretador.

43. (FCC/TRF-3ª Região - 2016) Um compilador

A) baseado em um sistema de compilação pura não realiza traduções; os programas são interpretados por um software interpretador.

B) do tipo JIT- Just in Time tem sido utilizado em linguagens como Java e C#. Na prática, JIT transforma um sistema híbrido em um sistema de compilação adiada.

C) que utiliza implementação híbrida traduz o programa para um código de formato intermediário e este código é compilado.

D) separa a tarefa de analisar a semântica do código em 2 partes: análise léxica e análise sintática. Este processo é realizado por uma máquina virtual.

E) de uma linguagem de programação traduz código de baixo nível para linguagem de máquina.

44. (FGV/IBGE - 2017) O módulo de análise léxica de um compilador tem por objetivo:

A) verificar se o programa-fonte obedece às regras da gramática da linguagem;

B) agrupar coerentemente os caracteres do programa-fonte em tokens;

C) gerar o código objeto correspondente à tradução do programa-fonte para alguma forma intermediária de representação;

D) construir as árvores sintáticas dos diversos comandos do programa-fonte;

E) eliminar comandos supérfluos do programa-fonte.

45. (FGV/MPE-AL - 2018) Na implementação de compiladores, a fase de parser do programa baseia-se, em parte, no resultado de um analisador léxico. Assinale a opção que descreve o papel de um analisador léxico.

A) Representar as regras da gramática da linguagem.

B) Verificar a conformidade do código fonte com as regras da gramática da linguagem.

C) Definir a notação em que as regras da gramática são expressas.

D) Identificar os tokens gramaticais no código fonte.

E) Exprimir a semântica das construções da linguagem.

46. (SUGEP-UFRPE/UFRPE - 2018) Em relação a compiladores, ligadores (link-editores) e interpretadores, é correto afirmar que:

A) uma função importante do tradutor é a realocação.

B) o link-editor é o utilitário responsável por carregar, na memória principal, um programa para ser executado.

C) o depurador é o utilitário responsável por gerar, a partir de um ou mais módulos-objeto, um único programa executável.

D) o link-editor é o utilitário responsável por gerar, a partir de um ou mais módulos-objeto, um único programa executável.

E) a grande vantagem do uso de interpretadores é o tempo gasto na tradução de suas instruções sempre que é executado.

47. (SUGEP-UFRPE/UFRPE - 2018) Abaixo, estão enumeradas as fases que integram o front-end de um compilador:

1) Análise Semântica

2) Análise Léxica

3) Análise Sintática

4) Gerador de código intermediário

Indique a sequência correta, com a ordem em que as fases ocorrem.

A) 1, 3, 2 e 4.

B) 3, 1, 4 e 2.

C) 2, 3, 1 e 4.

D) 1, 4, 2 e 3.

E) 4, 1, 2 e 3.

48. (IBADE/Câmara de Porto Velho-RO - 2018) Algumas linguagens exigem que o código fonte seja previamente traduzido para linguagem de máquina antes de ser executado. Chama-se esta fase de:

A) Linkedição.

B) Interpretação.

C) Tradução.

D) Compilação.

E) Edição.

49. (COMPERVE/UFRN - 2019) O conjunto de instruções de uma arquitetura de computador, ou Instruction Set Architecture - ISA, define as operações que podem ser utilizadas em um programa para ser executado nessa arquitetura. Um ISA define também

A) a linguagem de programação específica na qual o programa deve ser escrito.

B) o compilador específico que deve ser utilizado para compilar o programa.

C) o formato dos bits das instruções que são apresentadas ao processador.

D) a quantidade de estágios de pipeline que a arquitetura deve ter para executar o programa.

50. (CESPE/ANATEL - 2014) A compilação é o processo de análise de um programa escrito em linguagem de alto nível, denominado programa-fonte, e sua conversão em um programa equivalente, escrito em linguagem binária de máquina, denominado programa-objeto.

51. (CESPE/TCE-PA - 2016) Na compilação de um programa que chama uma biblioteca compartilhada, todo o código da biblioteca é copiado e inserido dentro do binário final.

52. (CESPE/TCE-PA - 2016) Os interpretadores, em vez de produzirem um programa objeto, fruto da tradução, executam diretamente as operações especificadas no código-fonte.

53. (CESPE/TCE-PA - 2016) Compilador é um utilitário responsável por executar um programa objeto diretamente na máquina.

54. (CESPE/ABIN - 2018) Chama-se cruzado o compilador que gera um programa que seja executável em pelo menos um sistema operacional diferente daquele onde o compilador tenha sido executado.

55. (CESPE/ABIN - 2018) Em um compilador, os tokens são identificados na fase de análise léxica e são representados por três propriedades: classe, valor e posição.

56. (CESPE/ABIN - 2018) A etapa de análise semântica de um compilador tem como objetivo verificar os inter-relacionamentos de um programa, validando tipologias, fluxos de controle e unicidade na declaração de variáveis.

57. (CESPE/EBSERH - 2018) Compilador é o programa que traduz o código fonte de uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível.

GABARITO

1. E
2. D
3. Certo
4. C
5. D
6. E
7. C
8. B
9. A
10. Errado
11. C
12. B
13. E
14. C
15. Errado
16. B
17. C
18. Errada
19. D
20. A
21. Errado
22. Errado
23. Certo
24. A
25. Certo
26. Certo
27. A
28. B
29. Errado
30. Certo
31. D
32. A
33. Certo
34. D
35. C
36. B
37. B
38. D
39. E
40. C
41. A
42. D
43. B
44. B
45. D
46. D
47. C
48. D
49. C
50. Certo
51. Errado
52. Certo
53. Errado
54. Certo
55. Certo
56. Certo
57. Certo

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.